

This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

**As rescanning documents *will not* correct images,
please do not report the images to the
Image Problems Mailbox.**

THIS PAGE BLANK (USPTO)

CORRECTED VERSION

(19) World Intellectual Property Organization
International Bureau



(43) International Publication Date
15 June 2000 (15.06.2000)

PCT

(10) International Publication Number
WO 00/034852 A1

(51) International Patent Classification⁶: G06F 3/14
(21) International Application Number: PCT/US99/29368
(22) International Filing Date:
10 December 1999 (10.12.1999)
(25) Filing Language: English
(26) Publication Language: English
(30) Priority Data:
09/209,162 10 December 1998 (10.12.1998) US
(71) Applicant: INTELLINET, INC. [US/US]; 325 Riverside
Drive, New York, NY 10025 (US).

(81) Designated States (*national*): AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States (*regional*): ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

(72) Inventor: BAKER, Michelle; 325 Riverside Drive, New York, NY 10025 (US).

Published:

— with international search report

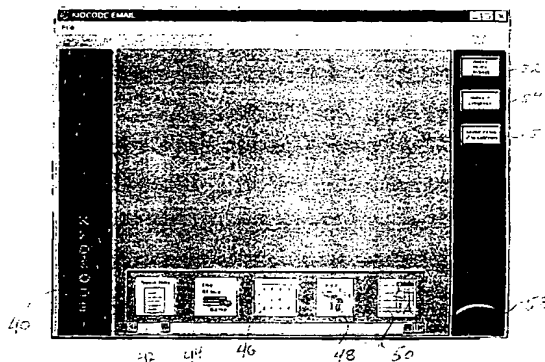
(74) Agent: GORDON, David, P.; 65 Woods End Road, Stamford, CT 06905 (US).

(48) Date of publication of this corrected version:

29 August 2002

[Continued on next page]

(54) Title: AUTHORIZING E-MAIL WITH MODULAR COMPONENTS



(57) Abstract: Electronic mail software includes a main email component (40) and a number installable components (42, 44, 46, 48, 50). The installable components (42, 44, 46, 48, 50) include authoring/reading components for creating/reading different kinds of documents and mailbox components for listing different kinds of messages or for listing messages in different styles. The main email component (40) provides an underlying graphical user interface for functions directly associated with the storage and transfer of electronic mail messages, and also handles all data bundling and unbundling required to transform a message created by an authoring component into a MIME compliant message. The authoring/reading components (42, 44, 46, 48, 50) act like applications embedded within the email program and allow specific types of documents such as spreadsheets, graphics, databases, etc. to be created from within the email program and emailed directly. The authoring/reading components (42, 44, 46, 48, 50) also allow received documents to be read without the difficulties traditionally associated with attaching binary files to an email letter. The authoring components of the invention pass data to the main email component (40) which packages the data as a MIME compliant message. When the message is received, the main email component concatenates and decodes the MIME message and sends the data to the authoring/reading component associated with the MIME type.

WO 00/034852 A1

**(15) Information about Correction:**

see PCT Gazette No. 35/2002 of 29 August 2002, Section II

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

AUTHORING E-MAIL WITH A MODULAR COMPONENTS

BACKGROUND OF THE INVENTION

1. Field of the Invention

The invention relates to an electronic mail program. More particularly, the invention relates to an electronic mail program having modular integral authoring/reading applications whereby documents created with the modular integral authoring/reading applications are seamlessly sent and received by the mail program.

2. State of the Art

In recent years electronic mail ("email") has become widely used in business, education, and in personal communications. One of the features of electronic mail which is most convenient, particularly in business and in education, is the ability to attach a binary computer file to an email message. This feature enables email correspondents to rapidly share word processing documents, database documents, spreadsheet documents, multimedia documents, or virtually any kind of binary file created by a computer. There are, however, some serious limitations and inconveniences associated with attaching a binary file to an email message.

The original Internet mail system as defined in 1982 with RFC (Request for Comments) 821 and 822 had a number of important limitations. In particular, the system was not designed to carry large quantities of arbitrary data in an email message. In fact, the 1982 SMTP (Simple Mail Transport Protocol) standard required that an email message consist of a single message containing only ASCII characters in lines of 1000 characters (blocks of 32k) or less. Moreover, some implementations of SMTP or other mail transport systems (such as UUCP) restricted message lengths to some allowed maximum number of bytes. Messages passing through a mail gateway using one of these implementations were likely to be truncated.

The ability to send large quantities of binary data through the Internet electronic mail system was made possible with the MIME (Multipurpose Internet Mail Extensions) standard for Internet messages. The original MIME standard was published as an Internet Request For Comments document (RFC 1341) and approved in June of 1992. (See Internet RFCs 2045, 2046, and 2047 for the latest MIME standards documents.) The MIME standard describes how an email message should be formatted in order to be considered MIME compliant. MIME defines a set of message header fields and a set of message encoding standards that are designed to overcome the limitations of RFC 822 message formats and still be transportable through any of the numerous legacy mail transport systems in use on the Internet.

MIME message header fields extend those defined in RFC 822 and describe the content and encoding type of the email message. Encoding schemes allowed in the MIME standard include "quoted-printable", and "base64". In addition, three unencoded data types are allowed. These are labeled "8bit", "7bit", or "binary".

If the sender and the receiver of the email message with the attached binary file are using the same brand and version of email program and both programs are configured in substantially the same way, the receiver's email program should automatically apply the appropriate decoding to the attached binary file and produce a file which is identical to the file which was attached to the email by the sender. However, if the sender and receiver are using different email programs, the recipient may receive a file which must be decoded by the recipient using a separate decoding program. Worse yet, if there is a failure of the receiving email program to properly deal with the MIME protocol, it is possible that the receiver will receive multiple files (each being $\leq 32k$) which must first be concatenated and then decoded.

Even after the file is properly received and decoded, it is often difficult for the receiver of the file to open the file. The receiver of the file might expect that "clicking" on the file icon will open the file. However, clicking on the file icon will often not open the file. It may result in an error message like "application not found" or, worse, it may result in the file being opened by an inappropriate application thereby displaying "gibberish". The receiver of the file must have a program capable of reading (opening) the file. For example, if one attaches a spreadsheet file to an email message, the receiver of the file must have a spreadsheet program in order to open the file. Technically, it is not necessary that the receiver of the file have the same brand program as that which created the file. However, opening a file with a program which did not create it, though possible, can be very inconvenient. The receiver of the file must know what kind of file is attached to the email message, must know what program on their computer is capable of reading that type of file, must launch the program, must open the file from within the program, and wait while the program translates the file.

The limitations of Internet electronic mail can become even more frustrating if the sender and recipient are not using the same operating system (OS). Some mail attachment encoding schemes (and file compression schemes) are OS-dependent and it is possible that an email recipient could receive a file which is impossible to decode (or decompress).

These limitations in electronic mail have discouraged many people, particularly non-sophisticated computer users, from attaching files to electronic mail messages. In fact, for some novice users, the task of launching one application to create a document, saving the document,

launching a separate email application to create an email message, and then locating the saved document for attachment to an email message is daunting enough to discourage them. In addition, novice users often complain that after "downloading" a file attached to an email message they cannot find the file on their hard disk.

SUMMARY OF THE INVENTION

It is therefore an object of the invention to provide an electronic mail program which includes integrated authoring software whereby a document may be created and sent by email in a seamless manner.

It is also an object of the invention to provide an electronic mail program which includes integrated authoring/reading software whereby a document may be received and opened in a seamless manner.

It is another object of the invention to provide an electronic mail program which includes modular integrated authoring software whereby different kinds of documents may be created and sent by email in a seamless manner.

It is still another object of the invention to provide an electronic mail program which includes modular integrated authoring/reading software whereby different kinds of documents may be received and opened in a seamless manner.

It is another object of the invention to provide an electronic mail program which includes modular integrated authoring/reading software whereby the authoring/reading software and the email software present an interface which suggests that a single application is operating.

It is another object of the invention to provide an electronic mail program which includes modular integrated mailbox handling software whereby messages of different types are displayed in different ways in a mailbox listing.

It is still another object of the invention to provide an electronic mail program which includes modular integrated authoring/reading software wherein the functionality of the authoring/reading software is controlled by the "role" of the user when participating in an exchange of messages.

In accord with these objects which will be discussed in detail below, the electronic mail software of the present invention includes a main email component and a number of installable components which communicate bidirectionally with the email component. The installable components include authoring/reading components as well as at least one mailbox browser/editor component. The main email component provides an underlying graphical user interface (GUI) for functions directly associated with the storage and transfer of electronic mail messages. In particular, the main email component provides menu items which allow the user to SEND, READ, REPLY, FORWARD, DELETE, SAVE, PRINT, for example. The main email program also handles all data bundling and unbundling that may be required to transform a message created by an authoring component into a fully MIME compliant message. In addition, the main email component includes "hooks" (an application programming interface or API) for the attachment of the installable components. The authoring/reading components each provide functionality which is particular to the type of document the component is designed to create/display. For example, a text document authoring component includes word processing functionality such as font selection, margin setting, etc. A painting/drawing authoring component includes tools for line drawing, polygon creation, paint brush, paint can, eraser, etc. A spreadsheet authoring component displays a grid and includes formula creation tools as well as formatting tools. A database authoring tool includes tools for creating fields and records, for sorting and searching, for generating reports, etc. A photo editor authoring component includes various imaging editing tools including cropping tools, dodging and burning tools, filters, etc. A presentation authoring component includes tools for creating slides and slide shows. The authoring components act like applications embedded within the email program and allow specific types of documents such as spreadsheets, graphics, databases, etc. to be created from within the email program and emailed directly. In addition, the authoring components allow received spreadsheets, graphics, databases, etc. to be read by the email program without the difficulties traditionally associated with attaching binary files to an email letter. According to the invention, in lieu of authoring components which allow both authoring and reading, separate components may be provided for authoring and reading, or components for reading only may be provided in addition to components which permit authoring as well as reading. The authoring/reading components interface with the main email component via designated "MIME types". The MIME data standard allows developers to define MIME types using the label "/application-x" in the data header. The authoring components of the invention pass data to the main email component which packages the data as a MIME compliant message with the label "/application-x" in the message header, where x identifies the authoring/reading component which created/can display the message. When the message is received, the main email component concatenates and decodes the MIME message, reads the MIME type, sends the data to the component associated with the MIME type, and waits for a user event or a callback from

the component. This bidirectional communication between the main email component and the authoring/reading components provides a totally seamless operation wherein the user may send and receive complex documents without any knowledge of attaching files, downloading, decoding, etc.

The mailbox browser/editor (mailbox handler) component is provided preferably as a separate component rather than as part of the main email component so that the software may be more easily customized and upgraded. The mailbox browser/editor component is used to display, edit, and browse mailboxes. Since the invention provides for email messages which contain different kinds of data, the features of the mailbox browser may depend on the type of messages being sent and received. For example, if a graphical authoring/reading component is installed, it may be desirable to provide a mailbox browser which shows a thumbnail of the selected graphic email message when a list of messages is displayed.

The software according to the invention provides a single seamless environment for authoring, reading, and emailing a variety of different types of documents. The user does not need to understand uploading, downloading, file types, file decoding, or any of the other esoteric requirements of attaching files to email. Further, the user does not need to know what kind of application must be launched in order to read a particular type of email message.

An exemplary embodiment of the invention is an email program for school children called KIDCODE®. The KIDCODE® program includes a main email component, a mailbox browser/editor component and several message authoring/reading components. The main email component and the mailbox browser/editor component provide the same functionality as described above. Additional KIDCODE® components include a text authoring tool, rebus game message handler components (encoding and decoding components) which allow children to create and respond to graphical rebus messages, several different game puzzle components, and a workbook which allows a teacher to send workbook problems to a student and allows the student to send the solved problems back to the teacher. According to one inventive aspect of the invention which is exemplified in the workbook and rebus components, an authoring/reading component may assign and track user "roles" by associating a role tag to each message. For example, in the rebus component, the user initiating the rebus exchange will be assigned the role of rebus encoder. The message created by this user will contain a tag identifying it as an "encoded message". When the message is opened by the recipient, the tools available in the rebus component will be different from those available if a message were being encoded. Similarly, the workbook component is preferably provided with a teacher role and a student role,

each of which have different tools. Component roles may be selected by the users, assigned by the system administrator, or automatically by components when messages are created/read.

According to a presently preferred embodiment, the KIDCODE® client software is written in the MACROMEDIA DIRECTOR™ LINGO™ scripting language which is cross-platform and thus ideally suited for use in schools which typically have a combination of MAC/OS™ and WINDOWS™ computers. As implemented, the client software operates over a TCP/IP LAN which is the most common type of network used in schools today and is compatible with the Internet. According to a further implementation of the invention, KIDCODE® software permits messages to be sent via the Internet in MIME compliant format.

Additional objects and advantages of the invention will become apparent to those skilled in the art upon reference to the detailed description taken in conjunction with the provided figures.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a screen shot of the KIDCODE® client login screen;

Figure 1a is a simplified state diagram illustrating the entry from the login screen to the main email component of the KIDCODE® software;

Figure 2 is a screen shot of the KIDCODE® main email component screen showing a menu of the installed authoring/display components and the mailbox browser component;

Figure 2a is a simplified state diagram illustrating the entry from the main email component into the installed components;

Figure 3 is a screen shot of the KIDCODE® mailbox browser/editor component screen;

Figure 4 is a screen shot of the KIDCODE® text message authoring component screen;

Figure 5 is a screen shot of the KIDCODE® rebus authoring (encoding) component screen;

Figure 6 is a screen shot similar to Figure 5 illustrating a listbox of users on the network to whom mail may be sent;

Figure 7 is a screen shot similar to Figure 5 illustrating a rebus in the process of being coded by the user;

Figure 8 is a screen shot of the KIDCODE® rebus reading (decoding) component screen;

Figure 9 is a screen shot of the KIDCODE® workbook authoring component screen;

Figure 10 is a screen shot illustrating the main email component of a second embodiment of the invention;

Figure 11 is a screen shot illustrating a text authoring component in the second embodiment of the invention;

Figure 12 is a screen shot illustrating a painting/drawing authoring component in the second embodiment of the invention;

Figure 13 is a screen shot illustrating a spreadsheet authoring component in the second embodiment of the invention;

Figure 14 is a screen shot illustrating a database authoring component in the second embodiment of the invention;

Figure 15 is a screen shot illustrating a photo editor authoring component in the second embodiment of the invention;

Figure 16 is a screen shot illustrating a slide show authoring component in the second embodiment of the invention; and

Figure 17 is a screen shot illustrating a display-only component in the second embodiment of the invention.

BRIEF DESCRIPTION OF THE APPENDICES

Appendix A is the LINGO™ script implementation of the KIDCODE® main email component;

Appendix B is a description of the Application Programming Interface for the KIDCODE® main email component which enables installable components to operate with the main email component.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

As mentioned above, a presently implemented embodiment of the invention is realized utilizing the MACROMEDIA DIRECTOR LINGO™ scripting language. The DIRECTOR™ application was originally intended to be used for authoring interactive multimedia presentations. Therefore, many of the terms used in the LINGO language refer to multimedia objects such as movies, cast members, frames, and sprites. However, it has been found that the LINGO™ language can be used to author many different types of programs including programs which were not traditionally thought of as being multimedia presentation programs. The following description, therefore, of the presently implemented embodiment will be best understood by those familiar with the MACROMEDIA DIRECTOR LINGO™ scripting language. However, those skilled in the art will understand from the functional description which follows that the invention could be implemented in other languages such as C or C++, JAVA™, etc.

Referring now to Figures 1 and 1a, and with reference to Appendix A, the first screen 10 presented by the KIDCODE® program is preferably a login screen where the user enters his or her name and password. According to the presently preferred embodiment, the login name field 12 is implemented as a popup menu (or pull down list box) and the password field 14 is a standard text entry field. See, for example, lines 172-190 of Appendix A. Clicking on the login name field will make a list of names appear and allow the user to highlight one of the names using the mouse. After the user has selected a name and typed in a password, the Okay button 16 must be clicked, or the Return or Enter key may be pressed. See Appendix A, lines 796-846 and lines 879-899. At this screen 10, the only option available under the FILE menu is QUIT. According to the presently preferred embodiment, the usernames and passwords are stored in associative (two property) lists so that a password may be associated with a username and a username may be associated with a password. When the okay button is clicked, the software checks the validity of the username and password. The checking of the username and password is illustrated in the state diagram in Figure 1a. Starting at 20 in Figure 1a, if the user selects a username and clicks the okay button, the password field is checked at 22. If no password was entered, a popup message is displayed at 24 indicating to the user that a password must be entered and the system returns to start at 20 waiting for the user to click the okay button. If the user types in a password and clicks the okay button, the username field is checked at 26. If no

username was selected, a popup message is displayed at 24 indicating that a username must be selected and the system returns to start at 20 waiting for the user to click the okay button. If the user types in a password and selects a username, it is determined at 28 or 30 whether the username and password match, and if they do, the software enters to the main email component at 32. If the username and password do not match, a popup message is displayed at 34 indicating that the password entered is invalid for the username selected and the system returns to start at 20 waiting for the user to click the okay button. If the username and password are that of the system administrator, a special administration display will be shown in which usernames and passwords may be added/deleted to/from the system. See Appendix A lines 858-875, 900-1016, and 1123-1140.

Turning now to Figures 2 and 2a, once the user has selected a username and entered the correct password, the program displays the screen 40 shown in Figure 2 (Appendix A lines 851-855). This is the screen of the main email component with no other component selected. The screen 40 includes a scrollable collection of icons 42, 44, 46, 48, 50 and includes buttons 52, 54, 56 for mailbox access and button 58 to quit the program. The icons 42, 44, 46, 48, and 50 represent the installed authoring/reading components. As shown in Figure 2, the icons represent a text component 42, a rebus component 44, a "text in grid" component 46, a puzzle component 48, and a "connect the dots" component 50. As illustrated in Figure 2a, starting at 60, if the user clicks on the text icon, the program will, at 62, open the text authoring component with an empty message (Appendix A lines 1422-1438 and 1025-1054). Similarly, if the user clicks on the rebus icon, the program will, at 64, open the rebus authoring component with an empty message (Appendix A lines 1442-1458 and 1025-1054). The main email component will also, when an authoring component is opened, make the print, trash, and send buttons visible as these functions are served by the main email component as illustrated in Figure 2a (Appendix A lines 489-526 and 1456). Since the presently implemented example does not yet have all components complete, the launching of other components is shown generically at 66 in Figure 2a. When a component is launched, the main email program suspends execution at 68 awaiting any "call back" from the launched component. The API described in Appendix B includes a set of "call back" functions that can be used by an authoring or mailbox component to communicate with the main email component. See Appendix B pages 4-7 and Appendix A lines 557-744. Further, the main email component serves the email functions of inbox, outbox, and filing cabinet for messages that have not been sent. These functions are illustrated in Figure 2 as buttons 52, 54, 56 and in Figure 2a as routines 72, 74, 76.

For example, as shown in Figure 3, when the user enters the inbox, a window 78 and a button bar 80 are displayed. The mailbox component with the appropriate set of messages is

launched as shown in Appendix A lines 1533-1558 and 216-247). The window 78 displays a list of new email and the button bar 80 displays buttons for functions common to all components, i.e. buttons for reply 82, send 84, print 86, and trash 88. It will be understood that depending on whether the user is in an authoring or reading mode, either the reply button 82 or the send button 84 will be "grayed out" indicating that that option is not available. As shown in Figure 3, for example, the send button 84 is not available when looking at the mail inbox. It will also be appreciated that the buttons and icons from the previous screen (Figure 2) are no longer visible. As stated in Figure 2a, the user returns to the screen of Figure 2 when the window 78 is closed.

The presently implemented text authoring/reading component is illustrated in the authoring mode in Figure 4. The window 90 is similar to any email message authoring tool and includes fields for "to:" 92, "from:" 94, "date" 96, "subject" 98, and "message" 100. The "from:" field 94 and "date" field 96 are hidden in Figure 4 behind the scrollable list box 102. After the addressee is chosen from the list box 102, the box disappears and reveals the "from" and "date" fields. According to the presently preferred embodiment, the list box 102 lists the names of all of the users registered in the local email system. This is handled by a call to the API as indicated at Appendix A lines 726-731. Thus, this embodiment prevents users from sending a document to a recipient who is not registered with the system administrator. It also allows users of the system to address messages without typing the recipient's name. According to another embodiment of the invention, shown and described below with reference to Figure 10, users are permitted to send email to any internet address and a list box is optionally used to display an address book. As seen in Figure 3, the reply button 82 in the button bar 80 is grayed out because that function is not available when authoring a text message. This is accomplished in Appendix A at lines 1422-1438.

Figures 5-8 illustrate the presently implemented rebus component. The rebus component presents a window 104 which includes a "to:" field 106, a message area 108, a "T" button 110 for selecting a template sentence, an "S" button 112 for hiding/displaying symbols, a "-" button 114 for hiding/displaying guesses, and a scrollable area 116 to the right of the message area which displays sets of symbols to be used in coding a rebus. In addition, the rebus component displays several pull down menus which are particular to it. These include the template choices menu 118 and the symbol choices menu 120.

According to the presently implemented embodiment, the author of a rebus begins by selecting a template sentence from a selection of sentences which are capable of being expressed as a rebus using the symbol sets provided. The template selection may be made via the T button

110 or the pull down menu 118. When a template sentence is selected, a suggested set of symbols is displayed in the field 116. Different symbols may be viewed by selecting a symbol set from the Symbol Choices menu 120. As illustrated in Figure 8, symbols are grouped according to the kinds of words they symbolize such as "action symbols", "object symbols", "quantity symbols", etc. As with other components of the KIDCODE® program, and as shown in Figure 6, the "to:" field 106 presents a drop down or pop up list box through which the email is addressed by selecting a registered user.

The author of the rebus codes the template sentence by dragging symbols from the scrollable field 116 to the message area 108. This is best illustrated by Figure 7. Symbols, when placed on a coded word in the template sentence, will snap into place when they are dragged into the area 108. According to the invention, not every word in the template sentence is designed to be coded with a symbol. According to the presently preferred embodiment, words which are to be coded appear in red text. For example, as shown in Figure 7, two symbols have been dragged into the message area and have snapped over the now hidden words "threw" and "window". The words "ball" and "through" are also red text and can be coded with a proper symbol. The author can hide the symbols and display the words in the sentence which are covered by symbols by clicking on the "S" button 112. However, when the recipient receives the rebus, clicking on the button "S" will not reveal the words beneath the symbols, but will only make the symbols disappear. This is an example of how user "roles" alter the tools available in a component. When the author finishes coding the rebus, he or she clicks on the send button 84. The main email component then automatically encodes the rebus as a MIME attachment to Internet mail and sends the mail to the recipient's mailbox.

Turning now to Figure 8, when the recipient of the rebus opens the email message containing a rebus, the KIDCODE® main email component automatically decodes the MIME attachment, determines that it is a rebus, and opens it in the rebus reading component. The message appears with empty text boxes (e.g. 122, 124, 126) beneath the graphic symbols. The recipient of the message must solve the rebus by typing in the text boxes the words which he/she believes are represented by the graphic symbols. As mentioned above, the "-" button 114 is for hiding/displaying the guesses typed in the boxes. When the recipient has typed in words for all the graphic symbols, he/she clicks on the reply button 82 to send the solution back to the author. Figure 8 shows the screen after the button 82 has been clicked. Thus it is grayed out to prevent the same message from being sent twice. The palette 116 is available to the decoder for browsing only. The features which allow symbols to be placed on the message are disabled for the decoder.

One of the authoring/reading components of the invention is a workbook message handler, an example of which is illustrated in Figure 9. The screen shot shown in Figure 9 illustrates the "student role" of a workbook message handler. The window 130 of the student role workbook message handler preferably includes "to:" and "from:" fields 132, 134 which are filled in by the teacher before the message is sent to the student, as well as six fields 136, 138, 140, 142, 144, 146 which must be filled in by the student before the message is returned to the teacher. As shown in Figure 9, the window 130 also includes a title 148, a date 150 and various instructions 152. Those skilled in the art will appreciate that the date 150 may be automatically entered when the message is sent to the student. The fields 136, 138, 140, 142, 144, 146, the title 148, and the instructions 152 may be manually entered by the teacher or may be selected as part of a template. In other words, the workbook message handler component may be a complex tool which allows teachers to author an infinite number of "problem messages" to students or it may be a modular set of pre-written problems or templates for problems. The workbook message handler component preferably includes many pre-written problems. Additional pre-written problems will be available through additional modular components. One important feature of the workbook message handler components is that they identify user status and automatically present the proper "role" of either teacher or student.

The KIDCODE® program described above is designed to be easy to use by very young children. Figures 10-17 illustrate another embodiment of the invention which is designed for a more sophisticated user, an older child, or an adult. The interface is more complicated, but offers the user more features. Referring now to Figure 10, the interface of the second embodiment of the invention includes a menubar 200 which lists some standard menus like FILE, EDIT, SETUP, WINDOW, and HELP. The menubar 200 may also include a clock display 202 which is typically supplied by the operating system and a blinking icon 204 which is typically supplied by TCP/IP connection software such as PPP dialup software, to indicate that the computer is connected to the Internet. The menus MAIL and FOLDER in the menubar 200 are particular to the emailing program and the scroll icon 206 is provided by scripting software, typically part of the operating system. Under the MAIL menu, one will find commands such as NEW for creating new mail, REPLY for replying to a mail message being read, FORWARD, etc. Under the FOLDER menu, one will find the names of user created filing cabinets (folders) where incoming mail can be saved. Under the SETUP menu, one will find commands for setting the necessary information to make a connection with the Internet, for storing the user's name and password, for scheduling automatic sending and receiving of mail, for performing automated tasks in response to mail (e.g. for automatically filing certain mail in certain folders, autoresponding to certain mail, etc.), etc. Under the WINDOW menu, the user will have the option of viewing INBOX, OUTBOX, FILING CABINET(s), CONNECTION STATUS, etc.

The HELP menu preferably provides a context sensitive alphabetical list of help items which are hot linked to html files.

Figure 10 illustrates the email program with a new outgoing message window 208 opened. The message window includes a standard text message field 210, a standard subject field 212, standard multiple recipient address fields 214, and a variety of buttons. The arrow buttons 216 allow the user to scroll among messages in the outbox. The outbox button 218 drops down a list of items in the outbox, from which items may be selected. The magnifying glass button 220 is used to search the user's address book. The "+" button 222 adds a recipient to the address field 214. The trash button 224 places the outgoing message in the trash and closes the window 208. The clock button 226 brings up a menu to schedule when the message will be sent. The rotating arrow button 227 causes the address fields 214 to disappear/reappear thereby expanding/contracting the size of the message field 210. The send button 228 sends the message to the outbox (if it is scheduled for delivery at another time or if the computer is not connected to the Internet) and sends the message otherwise. The button 230 labelled "-^K" causes the computer to connect to the Internet. As shown in Figure 10, this button 230 is grayed out because, as indicated by the blinking telephone pole icon 204, the computer is already connected to the Internet. The "send via" button 232 allows the user to select from several usernames, email accounts, etc.

The outgoing message window 208 shown in Figure 10 allows the user to send standard Internet mail by typing a message in the window 210. However, according to the invention, the window 208 also includes buttons 234, 236, 238, 240, 242, and 244, each of which is linked to an installed authoring/reading component. As described above, the number and nature of the authoring/reading components is modularly changeable. In the example shown in Figure 10, six authoring/reading components are shown to be installed. In practice, more, fewer, and/or different components may be installed. The components shown and described herein are: a word processor authoring/reading component linked to the button 234, a painting/drawing authoring/reading component linked to the button 236, a spreadsheet authoring/reading component linked to the button 238, a database authoring/reading component linked to the button 240, an image editor authoring/reading component linked to the button 242, and a presentation authoring/reading component linked to the button 244.

Turning now to Figure 11, when the user clicks on the button 234, the word processor component is invoked and it causes a new menubar 250 and a ruler 252 to appear inside the message field 210 of the window 208. The word processor component allows sophisticated formatting of messages which would be impossible in a normal Internet email program. For

example, margins can be set using the ruler 252; fonts can be changed using the FONT menu from the menubar 250; tables can be created and inserted using the TABLE menu from the menubar 250. In general, the menubar 250 provides much or all of the functionality of a full featured word processor program. Those skilled in the art will appreciate that the word processor interface shown in Figure 11 is similar to the interface of Microsoft® Word® 98. It will be noted that the menubar 250 provides a separate HELP menu in addition to the HELP menu provided on the menubar 200. It will be appreciated that the HELP menu could be omitted from the menubar 250 and the help files for the word processor component could be accessed from the main HELP menu on the menubar 200. It will also be noted that when the word processor component is invoked, the button 234 is grayed.

After a user creates a message with the word processor component, the addressing and mailing procedure is the same as sending an ordinary email. There is no need to save a file, encode it, or attach it to an email message. The main email component of the invention seamlessly performs all of the saving, encoding, and attaching without any of this being exposed to the user. More particularly, the authoring component and the main email component cooperate to save the authored document as a file on the user's disk. See Appendix B lines 229-238 and Appendix A lines 1293-1333 and 329-450. The main email component encodes the file in the MIME format with as many parts as necessary, and sends the MIME file(s) as Internet email message(s). When the message is received by a person using a copy of the email program of the invention, the receiver's main email component seamlessly concatenates the MIME parts, decodes the MIME file determines that it is a message created with the word processing component (Appendix A lines 690-694), invokes the word processing component (Appendix A lines 1019-1054), and opens the message with the word processing component (Appendix A lines 603-614). The receiver of the message does not have to download any file, find any attachment, execute any decoders, or launch any word processor to see the fully formatted document created by the sender.

Turning now to Figure 12, when the user clicks on the button 236, the painting/drawing component is invoked and it causes a new menubar 260 and a tool palette 262 to appear inside the message field 210 of the window 208. The painting/drawing component allows the author to create a painting (bitmap) graphic or a drawing (vectormap) graphic and send it to another user for viewing/editing. Those skilled in the art will appreciate that the menubar 260 and palette 262 shown in Figure 12 contain the menus and tool icons typically found in a full featured drawing/painting program. Those skilled in the art will appreciate that the painting/drawing component interface shown in Figure 12 is similar to the interface of Aldus® SuperPaint® 3.5. It will be noted that the menubar 260 provides a separate HELP menu in addition to the HELP

menu provided on the menubar 200. It will be appreciated that the HELP menu could be omitted from the menubar 260 and the help files for the painting/drawing component could be accessed from the main HELP menu on the menubar 200. It will also be noted that when the painting/drawing component is invoked, the button 236 is grayed.

After a user creates a graphic image with the painting/drawing component, the addressing and mailing procedure is the same as sending an ordinary email. There is no need to save a file, encode it, or attach it to an email message. The main email component of the invention seamlessly performs all of the saving, encoding, and attaching without any of this being exposed to the user. See Appendices A, E and F. When the message is received by a person using a copy of the email program of the invention, the receiver's main email component seamlessly concatenates MIME parts, decodes the MIME file, determines that it is a message created with the painting/drawing component, invokes the painting/drawing component, and opens the message with the painting/drawing component. The receiver of the message does not have to download any file, find any attachment, execute any decoders, or launch any painting/drawing program to view/edit the graphic image created by the sender. See Appendices A, E and F.

Turning now to Figure 13, when the user clicks on the button 238, the spreadsheet component is invoked and it causes a new menubar 270, a grid 272, and a tool palette 274 to appear inside the message field 210 of the window 208. The spreadsheet component allows the author to create a spreadsheet and send it to another user for viewing/editing. Those skilled in the art will appreciate that the menubar 270 and palette 274 shown in Figure 13 contain the menus and tool icons typically found in a full featured spreadsheet program. Those skilled in the art will appreciate that the interface of the spreadsheet component shown in Figure 13 is similar to the interface of Microsoft® Excel® 98. It will be noted that the menubar 270 provides a separate HELP menu in addition to the HELP menu provided on the menubar 200. It will be appreciated that the HELP menu could be omitted from the menubar 270 and the help files for the spreadsheet component could be accessed from the main HELP menu on the menubar 200. It will also be noted that when the spreadsheet component is invoked, the button 238 is grayed.

After a user creates a spreadsheet with the spreadsheet component, the addressing and mailing procedure is the same as sending an ordinary email. There is no need to save a file, encode it, or attach it to an email message. The main email component of the invention seamlessly performs all of the saving, encoding, and attaching without any of this being exposed to the user. See Appendices A, E and F. When the message is received by a person using a copy of the email program of the invention, the receiver's main email component seamlessly concatenates MIME parts, decodes the MIME file, determines that it is a message created with

the spreadsheet component, invokes the spreadsheet component, and opens the message with the spreadsheet component. The receiver of the message does not have to download any file, find any attachment, execute any decoders, or launch any spreadsheet program to view/edit the spreadsheet created by the sender. See Appendices A, E and F.

Turning now to Figure 14, when the user clicks on the button 240, the database component is invoked and it causes a new menubar 280, a record selection tool 282, and a free form space 284 to appear inside the message field 210 of the window 208. The database component allows the author to create a database and one or more reports and forms associated with the database and send it to another user for viewing/editing. Those skilled in the art will appreciate that the button bar 286 and the data fields 288 are defined by the author of the database using authoring tools found in the menus of the menubar 280. In fact, those skilled in the art will appreciate that the database interface shown in Figure 14 is similar to the interface of Filemaker®Pro 3.0. It will be noted that the menubar 280 provides a separate HELP menu in addition to the HELP menu provided on the menubar 200. It will be appreciated that the HELP menu could be omitted from the menubar 280 and the help files for the database component could be accessed from the main HELP menu on the menubar 200. It will also be noted that when the database component is invoked, the button 240 is grayed.

After a user creates a database with the database component, the addressing and mailing procedure is the same as sending an ordinary email. There is no need to save a file, encode it, or attach it to an email message. The main email component of the invention seamlessly performs all of the saving, encoding, and attaching without any of this being exposed to the user. See Appendices A, E, and F. When the message is received by a person using a copy of the email program of the invention, the receiver's main email component seamlessly concatenates MIME parts, decodes the MIME file, determines that it is a message created with the database component, invokes the database component, and opens the message with the database component. The receiver of the message does not have to download any file, find any attachment, execute any decoders, or launch any database program to view/edit the database created by the sender. See Appendices A, E, and F.

Turning now to Figure 15, when the user clicks on the button 242, the image editing component is invoked and it causes a new menubar 290 and a floating tool palette 292 to appear inside the message field 210 of the window 208. The image editing component allows the author to edit an image and send it to another user for viewing and/or further editing. Those skilled in the art will appreciate that the menubar 290 and palette 292 shown in Figure 15 contain the menus and tool icons typically found in a full featured image editing program. Those skilled

in the art will appreciate that the interface of the image editing component shown in Figure 15 is similar to the interface of Adobe® Photoshop® 3.5. It will be noted that the menubar 290 provides a separate HELP menu in addition to the HELP menu provided on the menubar 200. It will be appreciated that the HELP menu could be omitted from the menubar 290 and the help files for the database component could be accessed from the main HELP menu on the menubar 200. It will also be noted that when the database component is invoked, the button 242 is grayed. Those skilled in the art will appreciate that image editing software is typically not used to create an image but to edit an image created by some other hardware/software such as a digital camera or a scanner. As such, there is typically a menu item for opening or capturing an image. As shown in Figure 15, open/capture commands may be found under the FILE menu in the menubar 200. Alternatively, image acquisition commands may be found under a menu item in the menubar 290.

After a user edits an image with the image editor component, the addressing and mailing procedure is the same as sending an ordinary email. There is no need to save a file, encode it, or attach it to an email message. The main email component of the invention seamlessly performs all of the saving, encoding, and attaching without any of this being exposed to the user. See Appendices A, E, and F. When the message is received by a person using a copy of the email program of the invention, the receiver's main email component seamlessly concatenates MIME parts, decodes the MIME file, determines that it is a message created with the image editor component, invokes the image editor component, and opens the message with the image editor component. The receiver of the message does not have to download any file, find any attachment, execute any decoders, or launch any image editor program to view/edit the image edited by the sender. See Appendices A, E, and F.

Turning now to Figure 16, when the user clicks on the button 244, the presentation (slide show) component is invoked and it causes a new menubar 300, a floating wizard palette 302, and a blank template 304 to appear inside the message field 210 of the window 208. The presentation component allows the author to create a slide show presentation and send it to another user for viewing and/or editing. Those skilled in the art will appreciate that the menubar 300, palette 302, and template 304 shown in Figure 16 are typical of those found in a full featured presentation program. In fact, those skilled in the art will appreciate that the interface of the presentation component shown in Figure 16 is similar to the interface of Microsoft® PowerPoint® 98. It will be noted that the menubar 300 provides a separate HELP menu in addition to the HELP menu provided on the menubar 200. It will be appreciated that the HELP menu could be omitted from the menubar 290 and the help files for the database component

could be accessed from the main HELP menu on the menubar 200. It will also be noted that when the database component is invoked, the button 244 is grayed.

After a user creates a presentation with the presentation component, the addressing and mailing procedure is the same as sending an ordinary email. There is no need to save a file, encode it, or attach it to an email message. The main email component of the invention seamlessly performs all of the saving, encoding, and attaching without any of this being exposed to the user. See Appendices A, E, and F. When the message is received by a person using a copy of the email program of the invention, the receiver's main email component seamlessly concatenates MIME parts, decodes the MIME file, determines that it is a message created with the presentation component, invokes the presentation component, and opens the message with the presentation component. The receiver of the message does not have to download any file, find any attachment, execute any decoders, or launch any presentation program to view/edit the presentation created by the sender. See Appendices A, E, and F.

As described above, messages received by the email software according to the invention are seamlessly decoded and displayed. Figure 17 illustrates an incoming message window 408 which displays a message containing a combination of text and graphics in the message field 410. The incoming message window 408 also includes a subject field 412 and a "from:" address field 414 which includes information about the time the message was sent and received. Arrow buttons 416 allow the user to scroll through messages in the "in box". Button 418 drops a menu list of messages in the in box from which a message may be selected. The "+" button 420 adds the sender's address to the recipient's address book. The rotating arrow 427 hides the address field 414 and expands the message field 410. Buttons 428 and 430 are not implemented, but may be used for public key decryption, etc.

As mentioned above, the modular components of the invention may be authoring/reading components or read only components. Figure 17 illustrates an incoming message window 408 which displays a message containing a combination of text and graphics in message field 410 without any editing/authoring tools. The message may have been created with the word processing component or the painting and drawing component. The component used to create the message need not be known by the recipient of the message when it is opened with a read only component as shown. It will be appreciated that the message could also be automatically opened with an authoring/reading component, in which case, the message field 410 in Figure 17 would also include a menubar, and perhaps a tool palette. According to the invention, the email client software may be provided with a full complement of read only components and the authoring components may be installed according to the user's choices. Additionally, the email

client software may be programmed to automatically download a reading component from an ftp site when it encounters a message which requires a component which is not yet installed.

There have been described and illustrated herein several embodiments of electronic mail software with modular integrated authoring/reading software components. While particular embodiments of the invention have been described, it is not intended that the invention be limited thereto, as it is intended that the invention be as broad in scope as the art will allow and that the specification be read likewise. Thus, while particular graphical interfaces have been disclosed, it will be appreciated that other interfaces could be utilized. Also, while particular authoring/reading components have been shown, it will be recognized that other types of authoring/reading components could be provided in the spirit of the invention. Moreover, while particular configurations have been disclosed in reference to the code in the appendices, it will be appreciated that other configurations could be used as well. Further, while particular software code and pseudocode have been disclosed to perform various functions, it will be appreciated that other code and/or hardware could be utilized to accomplish those functions and should be considered the equivalents thereof. It will therefore be appreciated by those skilled in the art that yet other modifications could be made to the provided invention without deviating from its spirit and scope as so claimed.

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```

1  on startMovie
2
3  global emG_passwordList, emG_userGroupList, emG_userGroup, emG_userName,
4  emG_registeredUsers, emG_msgNumber, emG_maildata, emG_mode, emG_noSimulate,
5  emG_mailFileList, emG_boxName
6
7  --- Register the "YAK YAK" text to speech xtra
8  -- register xtra "Yak", "XXXXXXXXXXXXXXX"
9
10 -- VARIABLE LIST
11 --- emG_userName: Tracks current user by name
12 --- emG_msgNumber: Tracks if a message is new (empty) or old (number)
13 --- emG_registeredUsers: Tracks users for to boxes in movies
14 --- emG_passwordList: List of passwords for user login: [password:name]
15 --- emG_maildata: Message data list:
16 ---   #to, #from, #re, #date, #mimetype, #mbxName, #msgbody
17 -- NOT IMPLEMENTED -> #mbxName: now takes the place of #status - eliminate
18 case statement...
19 --- emG_mode: flag for message movies; #author, #display
20 --- emG_noSimulate: disable simulate Mode for message handler movies
21 --- emG_userGroupList: for testing rebus game
22 --- emG_userGroup: for testing rebus
23 --- emG_mailFileList: List of locations of mailfiles for each user:
24 ---   [uname:filename]
25 --- emG_boxName: a mailbox datastructure; used to pass mailboxes to the mailbox
26 movie
27
28 -- Install the menu
29 installMenu "main menu"
30
31 -- Clear all global variables
32
33 set emG_noSimulate = TRUE
34
35 --- Make sure the AddUsers button is not visible
36 set the visible of sprite 20 = FALSE
37
38 initSystemUsersData()
39 initializeUser()
40 initializeFields()
41 fillStudentName()
42 clearPassword()
43
44 end
45
46
47 on stopMovie
48
49 global instanceOfXtra, emG_passwordList, emG_userGroupList, emG_userGroup,
50 emG_userName, emG_msgNumber, emG_maildata, emG_mode
51
52 -- Clear all fields and global variables
53
54 put "" into field "addPass"
55 put "" into field "addUserGroup"
56 put "" into field "addName"
57 put "" into field "userList"
58 put "" into field "studentName"

```


Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```

59 put "" into field "studentUpName"
60 put "" into field "studentPass"
61
62 put "" into emG_userName
63 set emG_msgNumber = 0
64 set emG_registeredUsers = []
65 set emG_passwordList = [:]
66 set emG_userGroupList = [:]
67 set emG_maildata = [:]
68
69 set emG_userGroup = 0
70 set emG_mode = #empty
71
72 clearPassword()
73
74 -- empty the script used to read in mailboxes
75 set the scriptText of member 65 = ""
76
77 --- Make sure the AddUsers button is not visible
78 set the visible of sprite 20 = FALSE
79
80 end
81
82
83 -- score script 3 ss_goTheFrame
84
85 on exitFrame
86
87 go the frame
88
89 end
90
91
92 --- Modified 8-9-98. To include a mailfile location for each
93 --- user. Added global variable emG_mailFileList. Also changed
94 --- format of the users file to be comma delimited items. This
95 --- will avoid problem with spaces in full pathnames for
96 --- user mailbox files.
97
98 on initSystemUsersData
99 global emG_registeredUsers
100 global emG_passwordList, emG_userGroupList, emG_mailFileList
101
102 set emG_registeredUsers = []
103 set emG_passwordList = [:]
104 set emG_userGroupList = [:]
105 set emG_mailFileList = [:]
106
107 set userData = readUsersFile()
108
109 put the number of lines of userData into totalLines
110 repeat with i = 1 to totalLines
111
112 if line i of userData = EMPTY then
113 nothing
114 else
115 set uname = item 1 of line i of userData
116 set pw = item 2 of line i of userData

```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```

117    set ugroup = value(item 3 of line i of usersData)
118    set mfile = item 4 of line i of usersData
119
120    add emG_registeredUsers, uname
121    addProp emG_passwordList, uname, pw
122    addProp emG_userGroupList, uname, ugroup
123    addProp emG_mailFileList, uname, mfile
124    end if
125
126  end repeat
127
128  sortRegisteredUsers()
129
130 end initSystemUsersData
131
132 -----
133 on initializeUser
134
135   global emG_userGroup, emG_userName
136   global emG_msgNumber, emG_maildata, emG_mode
137
138   put "" into emG_userName
139   set emG_msgNumber = 0
140   set emG_maildata = [:]
141   set emG_userGroup = 0
142   set emG_mode = #empty
143
144 end initializeMyGlobals
145
146 -----
147 -- Initialize formatting of all visible text fields
148 -- Should be called when movie starts
149
150 on initializeFields
151
152   -- SetTextInfo "StudentName", " ", "left", "arial", 14, "bold"
153   SetTextInfo "StudentUpName", "your username here ", "left", "arial", 14, "bold"
154   SetTextInfo "StudentPass", "", "left", "arial", 14, "bold"
155
156   put "" into field "addPass"
157   put "" into field "addUserGroup"
158   put "" into field "addName"
159   put "" into field "userList"
160
161   -- set the lineHeight of field "To" = 18
162   -- set the border of member "To" = 1
163   -- set the border of member "ToDown" = 1
164   -- set the margin of member "ToDown" to 8
165
166 end initializeFields
167
168 -----
169 -- THIS HANDLER FILLS THE STUDENT LOGON NAME FIELD
170 -- WITH THE CURRENT LIST OF STUDENT NAMES
171
172 on fillStudentName
173   global emG_registeredUsers
174

```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```

175 -- Clear the student name field for the kids' logon
176 put "" & RETURN into field "studentName"
177
178 repeat with uname in emG_registeredUsers
179     put uname & RETURN after field "studentName"
180
181 end repeat
182
183 -- Bring the field back to the top line
184 set the scrollTop of member "studentName" = 0
185
186 end
187
188 -----
189
190 -- For convenience of all the message handling movies
191 -- keep emG_registeredUsers in a special sorted order:
192 -- alphabetic with "administrator" at the end.
193
194 on sortRegisteredUsers
195     global emG_registeredUsers
196
197     -- fix up emG_registeredUsers in sorted order but
198     -- with "administrator" at the end
199
200     deleteOne(emG_registeredUsers, "administrator")
201     sort(emG_registeredUsers)
202     append(emG_registeredUsers, "administrator")
203
204 end sortRegisteredUsers
205 --mailbox handlers
206
207 -----
208 --- openMailbox starts the mailbox movie
209 --- because the call must be continued in emh_continue
210 --- it is necessary to use a global variable for the
211 --- mailbox name.
212
213 on openMbx boxName
214     global emG_boxName
215
216     set emG_BoxName = boxName
217
218     go to frame "movie"
219
220     -- since all sprites are automatically puppets in Dir 6.0
221     -- next should not be necessary
222     -- Take control of the sidebar buttons
223
224     puppetSprite 6, TRUE
225     puppetSprite 7, TRUE
226     puppetSprite 8, TRUE
227     puppetSprite 9, TRUE
228
229     set mbxMovie = window "mailbox.dir"
230     set the titleVisible of mbxMovie to FALSE
231     set the rect of mbxMovie = getMovieRect("mailbox")
232

```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```

233 open mbxMovie
234 set the name of mbxMovie to "childWindow"
235
236 tell window "childWindow"
237 -- next is a hack to get around Macromedia MIAW bug
238 -- see emh_continue for calls to real handlers
239 emc_startMeUp()
240
241 end tell
242
243 -- CONTINUES in emh_continue
244 end
245
246
247 -- Read mailbox accepts a string that is the mailbox name
248 -- and returns a mailbox datastructure that is the
249 -- mailbox name and a list of the messages in that box
250
251 on readMailbox boxName
252   global emG_userName, emG_mailFileList
253   --
254   -- "inbox" : set bxstring = "#mbxName: #received"
255   -- "outbox" : set bxstring = "#mbxName: #sent"
256   -- "savebox" : set bxstring = "#mbxName: #saved"
257   -- "trashbox" : set bxstring = "#mbxName: #trashed"
258
259   set msgList = []
260
261
262   set mbxStruc = list(boxName, msgList)
263   set mailFileName = getProp(emG_mailFileList, emG_userName)
264
265   -- Start up Fileio Xtra
266   set instanceOfXtra = new(xtra "fileio")
267
268   -- Set up Fileio to read from users file
269   openFile(instanceOfXtra, mailFileName, 1)
270
271
272   -- If file users doesn't exist, create it and set it up for read
273   if status(instanceOfXtra) < 0 then
274     createFile(instanceOfXtra, mailFileName)
275     openFile(instanceOfXtra, mailFileName, 1)
276   end if
277
278   -- Read what's currently in the file
279   set whatText = readFile(instanceOfXtra)
280
281   -- put msgs from appropriate box into the message list
282   -- this needs to be fixed after the mail file datastructure
283   -- is changed...
284
285   -- if value(#mbxname) < 0 then
286   --   alert "Invalid mailbox name."
287   --   return(0)
288   -- end if
289
290   --OLD case statement

```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```

291 case boxname of
292     "inbox" : set bxstring = "#status: #received"
293     "outbox" : set bxstring = "#status: #sent"
294     "savebox" : set bxstring = "#status: #saved"
295     "trashbox" : set bxstring = "#status: #trashed"
296
297     otherwise:
298         alert "Invalid mailbox name."
299         return(0)
300 end case
301
302 -- inefficient to have to look for the "#status...string"
303 -- now is changed to value(#string) turning the string into a value, as
304 -- Director has difficulties with strings w/in property lists...
305
306 repeat with i = 1 to the number of lines in whatText
307     if line i of whatText contains bxstring then
308         append(msgList, value(line i of whatText))
309     end if
310 end repeat
311
312 -- Close Fileio Xtra
313
314 closeFile(instanceOfXtra)
315
316 set instanceOfXtra = 0
317
318 return(mbxStruc)
319
320 end
321
322 on messageHandler msgStatus
323
324     global emG_userName, emG_maildata, emG_msgNumber, emG_mode,
325     emG_mailFileList
326
327     put "" into sendData
328
329     setProp emG_maildata, #status, msgStatus
330
331
332     -- Set up where to find the users mailfile
333     set whatFile = getProp(emG_mailFileList, emG_userName)
334
335
336     -- Start up Fileio Xtra
337     set instanceOfXtra = new(xtra "fileio")
338
339
340     -- Set up Fileio to read and write from/to users file
341     openFile(instanceOfXtra, whatFile, 0)
342
343
344     -- If file users doesn't exist, create it and set it up for read/write
345     if status(instanceOfXtra) <> 0 then

```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
349   createFile(instanceOfXtra, whatFile)
350   openFile(instanceOfXtra, whatFile, 0)
351   end if
352
353
354   -- Read what's currently in the file
355   set whatText = readFile(instanceOfXtra)
356
357
358   -- Add message to current user's mailbox
359   --- if it previously existed, then write over the old message
360   --- if not, add it to the bottom
361   --- Only messages with a status = #saved can be changed.
362
363   if emG_msgNumber < 0 then
364     repeat with i = 1 to the number of lines in whatText
365       if i = emG_msgNumber then
366         put emG_maildata & RETURN after sendData
367       else if line i of whatText < "" then
368         put line i of whatText & RETURN after sendData
369       end if
370     end repeat
371
372   else if emG_msgNumber = 0 then
373     put whatText into sendData
374     put emG_maildata & RETURN after sendData
375   end if
376
377
378   -- Put the cursor at the begining of the users file
379   setPosition(instanceOfXtra, 0)
380
381
382   -- Overwrite users file with updated list
383   writeString(instanceOfXtra, sendData)
384
385
386   -- Close Fileio Xtra
387
388   closeFile(instanceOfXtra)
389
390   set instanceOfXtra = 0
391
392
393   -- ON SEND, PUT IN OTHER CHILD'S MAILBOX, TOO
394
395   if msgStatus = #sent then
396
397     setaProp emG_maildata, #status, #received
398
399     put getaProp(emG_maildata, #to) into sendingTo
400
401     put "" into sendData
402
403
404     -- Set up where to find the users file
405     -- put the pathName & sendingTo into whatFile
406     set whatFile = getProp(emG_mailFileList, sendingTo)
```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
407
408
409 -- Start up Fileio Xtra
410 set instanceOfXtra = new(xtra "fileio")
411
412
413 -- Set up Fileio to read and write from/to users file
414 openFile(instanceOfXtra, whatFile, 0)
415
416
417 -- If file users doesn't exist, create it and set it up for read/write
418 if status(instanceOfXtra) < 0 then
419   createFile(instanceOfXtra, whatFile)
420   openFile(instanceOfXtra, whatFile, 0)
421 end if
422
423
424 -- Read what's currently in the file
425 set whatText = readFile(instanceOfXtra)
426
427 -- Add message to recipient's mailbox
428 put emG_maildata & RETURN after whatText
429
430 -- Put the cursor at the beginning of the users file
431 setPosition(instanceOfXtra, 0)
432
433 -- Overwrite users file with updated list
434 writeString(instanceOfXtra, whatText)
435
436
437 -- Close Fileio Xtra
438
439 closeFile(instanceOfXtra)
440 set instanceOfXtra = 0
441
442 end if
443
444 end
445 -----
446 on createMailData userName, type
447
448   set newmsg = []
449   addProp(newmsg, #to, "")
450   addProp(newmsg, #from, userName)
451   addProp(newmsg, #re, "")
452   addProp(newmsg, #date, the abbreviated date)
453   addProp(newmsg, #mimetype, type)
454   addProp(newmsg, #status, #new)
455   addProp(newmsg, #msgbody, [])
456   return(newmsg)
457
458 end createMailData
459
460 -----
461 --- Make sure there is something in each of the "to"
462 --- and "from" fields and that the messagebody has the
463 --- right format.
464
```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
465 on isValidMessage maildata
466
467   repeat with prop in [#to, #from]
468     if getProp(maildata, prop) = "" then
469       alert "But who do you wish to send this message to?"
470       --return(0)
471     end if
472   end repeat
473
474   if not listp(getProp(mailData, #msgBody)) then return(0)
475
476   return(1)
477
478 end isValidMessage
479
480
481 -----
482
483 on setReply
484
485   -- TAKES CARE OF SWITCHING THE SIDEBAR BUTTONS WHEN REPLY
486   -- IS HIT FROM AN OPEN MESSAGE
487
488   go to "Movie" -- make sure the frame is correct
489
490   -- Set the buttons with reply off and send on
491   disableReply()
492   enableSend()
493
494 end
495
496 -----
497
498 on disableSend
499   go to "movie"
500   puppetsprite 7, TRUE
501   set the member of sprite 7 = member "SendNo"
502 end disableSend
503
504 on enableSend
505   go to "movie"
506   puppetsprite 7, TRUE
507   set the member of sprite 7 = member "Send"
508 end enableSend
509
510 on disableReply
511   go to "movie"
512   puppetsprite 6, TRUE
513   set the member of sprite 6 = member "ReplyNo"
514 end disableSend
515
516 on enableReply
517   go to "movie"
518   puppetsprite 6, TRUE
519   set the member of sprite 6 = member "Reply"
520 end enableSend
521
522
```


Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```

523 on returnToMain
524
525   global emG_msgNumber, emG_maildata, emG_mode
526
527   -- Clear the variables
528
529   set emG_msgNumber = 0
530   set emG_maildata = [:]
531   set emG_mode = #empty
532
533   --- unpuppet the left panel buttons which reuse sprite
534   --- channels 6-9
535   -- MB 10-13-98 I don't like this method... it is safer
536   -- to use new sprite channels.
537   -- is there a good reason for reusing channels...does it
538   -- affect performance?
539
540   puppetsprite 6, FALSE
541   puppetsprite 7, FALSE
542   puppetsprite 8, FALSE
543   puppetsprite 9, FALSE
544
545   -- Go back to the main menu
546
547   go to "open"
548
549 end
550
551 -- API handlers
552
553 --- emh_getUserMailbox returns the current user's mailbox specified
554 --- by the mailBoxName parameter.
555
556 on emh_getUserMailbox mailboxName
557
558   return(readMailbox(mailBoxName))
559
560 end emh_getUserMailbox
561
562 -----
563
564 on emh_getUserName
565   global emG_userName
566
567   return(emG_userName)
568
569 end emh_getUserName
570
571 -----
572
573 on emh_getUserData userName
574   global emG_userGroupList, emG_mailFileList
575
576   return(list (username, --
577     username, getProp(emG_userGroupList, username), getProp(emG_mailFileList,
578     userName), [], list ( "inbox", "outbox", "savebox" ) ))
579
580 end emh_getUserData

```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```

581
582
583 -- more API handlers
584
585 -----
586 --- A curse on Macromedia. This ugly hack is used to get
587 --- around a Macromedia bug which causes the startMovie
588 --- handler of a MIAW to run only after control has been
589 --- transferred back to the calling movie and the calling
590 --- movie advances a frame.
591
592 --- This handler is called by the startMovie handler of the
593 --- MIAW. This way we ensure that these scripts only run
594 --- after the MIAW has been properly initialized.
595
596 on emh_continue componentType
597   global emG_userName, emG_maildata, emG_mode, emG_boxName, emG_userGroup
598
599   -- Since this function can only be called by a MIAW component
600   -- we assume that the "childwindow" is running
601
602   if componentType = #msgHandler then
603     tell window "childwindow"
604       emc_initWindow(emG_userName)
605       msh_openMessage(emG_maildata, emG_mode)
606     end tell
607
608   else if componentType = #mailbox then
609     tell window "childwindow" to emc_initWindow(emG_userName)
610     set success = the result
611     if not success then
612       alert "Could not initialize mailbox movie"
613       forget window "childwindow"
614       return(0)
615     end if
616
617     set mbx = readMailbox(emG_boxName)
618     tell window "childwindow" to mbx_openMailbox(mbx)
619     set success = the result
620     if not success then
621       alert "Could not open mailbox."
622       forget window "childwindow"
623       return(0)
624     end if
625
626   else alert "ERROR invalid componenttype."
627
628 end emh_continue
629
630
631 -- more API handlers
632 -----
633 -- The emh_passMessage handler is used to pass a message from
634 -- a mailbox to the appropriate message handler
635
636 on emh_passMessage maildata, messageNumber
637
638   global emG_maildata, emG_msgNumber, emG_mode

```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```

639
640 -- should check for errors in the parameters
641
642 set emG_maildata = mailData
643 set emG_msgNumber = messageNumber
644
645 -- If a mailbox window is open we need to close that window.
646 -- The window will not actually close until this function completes
647 -- and returns control to the caller function in the mailbox movie.
648 -- Therefore, we need to move it to the back so it is no longer visible.
649
650 moveToback window "childwindow"
651 updatestage
652
653 tell window "childWindow" to emc_getComponentInfo()
654 set cInfo = the result
655 if getComponentProp(cInfo, #ComponentType) = #mailbox then
656     tell window "childWindow" to emc_closeWindow()
657     forget window "childWindow"
658 end if
659
660 go to frame "movie"
661 -- set up the button bar on the left
662
663 set msgStatus = getProp(emG_maildata, #status)
664 if msgStatus = #received then -- from inbox
665     set emG_mode = #display
666     disableSend()
667     enableReply()
668 else if msgStatus = #sent then -- from outbox
669     set emG_mode = #display
670     disableSend()
671     disableReply()
672 else if msgStatus = #saved then -- from savebox
673     set emG_mode = #author
674     disableReply()
675     enableSend()
676 else -- error
677     alert "passing message with invalid status"
678     return(0)
679 end if
680
681 --- OPEN MESSAGE HANDLER MOVIE
682
683 openMsgHandler(getaProp(emG_maildata, #mimetype), emG_maildata)
684
685 end emh_passMessage
686
687 -- more API handlers
688
689 -- THIS CODE IS BASED ON OLD STUFF WHICH USES MESSAGE NUMBER
690 -- TO IDENTIFY MESSAGES ACROSS MAILBOXES. THIS SYSTEM
691 -- NEEDS TO BE CHANGED TO IDENTIFY MESSAGES BY A MAILBOXNAME
692 -- AND A MESSAGE NUMBER WITHIN THE BOX
693
694 on emh_getMessage messageNumber, typeorBoxName
695
696     global emG_userName, emG_msgNumber, emG_mailData

```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```

697
698 set emG_msgNumber = messageNumber
699
700 if messageNumber = 0 then -- return new message data
701   --typeorBoxName should have mimetype
702   set emG_maildata = createMailData(emG_userName, typeorBoxName)
703   return(emG_maildata)
704 end if
705
706 -- otherwise find an existing message
707 -- typeorboxname should have boxName
708
709 set theBox = readMailbox(typeorBoxName)
710 set emG_mailData = getAt(getAt(theBox, 2), messageNumber)
711 return(emG_maildata)
712
713 end emh_getMessage
714
715 -----
716
717 on emh_getRegisteredUsers
718   global emG_registeredUsers
719
720   return(emG_registeredUsers)
721
722 end emh_getRegisteredUsers
723
724 -----
725
726 on emh_killComponent
727
728   tell window "childwindow" to emc_closeWindow()
729   if the result = 0 then alert "TROUBLE CLOSING WINDOW!"
730   else
731     forget window "childwindow"
732   end if
733   returnToMain()
734
735 end emh_killComponent
736
737 -----
738 --- Initialize formatting of text fields
739 --- Thanks to Frank Leahy, maricopa site for this one.
740
741 on SetTextInfo fldName, fldValue, fldAlign, fldFont, fldSize, fldStyle
742
743   put fldValue into field fldName
744   set the textAlign of field fldName = fldAlign
745   set the textFont of field fldName = "arial" --fldFont
746   set the textSize of field fldName = fldSize
747   set the textStyle of field fldName = fldStyle
748
749 end
750
751 -----
752
753 -- script of cast member studentName
754 -- emG_userName should not be set here

```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
755 -- because it could be invalid
756
757 on mouseUp
758
759 -- Put selected user name into up version of student field
760 -- switch the field from down to up
761
762 put word 1 of line(the mouseLine) of field "studentName" into field "studentUpName"
763
764 set the member of sprite 14 to member "StudentUpName"
765
766 end
767
768
769 -- script of cast member studentUpName
770
771 on mouseUp
772
773 -- Pull down student field: change field from
774 -- up (sprite 17) to down (sprite 16)
775
776 set the member of sprite 14 to member "StudentName"
777
778 -- clear password field
779 clearPassword()
780
781 end
782
783
784
785 -- scripts of cast member studentPassword
786
787
788 on keyUp
789 global gpw, gpwlen
790 --gpw is global password and
791 --gpwlen is global password length
792
793 hideAlert() -- user maybe trying again...hide badPwMsg
794
795 if the key = RETURN then
796 if checkPassword(field "studentUpName", gpw) then
797 enterMainEmail(field "studentUpName")
798 else --- invalid password
799 alertBadPassword()
800 end if
801 set gpw = ""
802 set gpwlen = 0
803 put "" into field "studentPass" -- reset the password field
804 end if
805
806 end keyUp
807
808
809
810 on keyDown
811 global gpwlen, gpw
812
```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
813 --eats the key, otherwise it will appear until keyup
814
815 if the key = BACKSPACE then
816     put "" into char gpwlen of field "studentPass"
817     put "" into char gpwlen of gpw
818     if gpwlen > 0 then
819         set gpwlen = gpwlen - 1
820     end if
821 else if the key = RETURN then
822     nothing
823 else if the keycode >= 117 and the keycode <= 126 then
824     nothing
825 else
826     put "*" after field "studentPass"
827     put the key after gpw
828     set gpwlen = gpwlen + 1
829
830 end if
831
832 set the selstart = gpwlen
833 set the selend = the selstart
834
835 end keyDown
836
837
838 -- script of cast member goStudentLog
839
840 on mouseUp
841     go to frame "pass"
842
843
844 end
845
846
847 -- script of cast member editUsers
848
849 on mouseUp
850
851     -- set the default pathname for the mail file location
852     put the pathname into field "addMailFileLoc"
853
854     go to frame "edit"
855
856 end
857
858
859 -- script of cast member okUser
860
861
862 on mouseDown
863     set the member of sprite 7 = "okay down"
864 end
865
866
867
868 -- script of cast member okDown
869
870
```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
871 on mouseUp
872   global gpw, gpwlen --- see script of field studentPass
873
874   set the member of sprite 7 = "okayUser"
875
876   if checkPassword(field "studentUpName", gpw) then
877     -- valid user & pw
878     enterMainEmail(field "studentUpName")
879
880   else -- password invalid
881
882     alertBadPassword()
883
884   end if
885
886   clearPassword()
887
888 end
889 -- script of cast member addUser
890
891 on mouseUp
892   global emG_registeredUsers
893   global emG_passwordList, emG_userGroupList, emG_mailFileList
894
895   --check that username is filled and is unique
896
897   if field "addName" = EMPTY then
898     alert "No username"
899     return(0)
900   else if getOne(emG_registeredUsers, field "addName") then
901     alert "Username already in system. Choose a different name"
902     return(0)
903
904   else set uname = field "addName"
905
906   --NEED TO TAKE CARE OF THIS!!!!
907   -- check that the mailfile location is a valid directory
908   -- there are serious problems with this at present
909   -- for now assume pathnames are valid
910
911   -- add new User data to system global variables
912   add(emG_registeredUsers, uname)
913   addProp(emG_passwordList, uname, field "addPass")
914   addProp(emG_userGroupList, uname, field "addUserGroup")
915   -- append username to the mailfile location directory
916   addProp(emG_mailFileList, uname, field "addMailFileLoc" & uname)
917
918   sortRegisteredUsers()
919
920   -- write the users file with system users data
921   writeUsersFile()
922
923   -- Put the updated user list into the userList field
924   put "" into field "userList"
925   repeat with uname in emG_registeredUsers
926     put uname after field "userList"
927     put " " & getProp(emG_passwordList, uname) after field "userList"
```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```

929   put "" & getProp(emG_userGroupList, uname) after field "userList"
930   put "" & getProp(emG_mailFileList, uname) after field "userList"
931   put RETURN after field "userList"
932 end repeat
933
934
935 -- reset the User data fields
936
937 put "" into field "addUserGroup"
938 put "" into field "addPass"
939 put "" into field "addName"
940 put the pathname into field "addMailFileLoc"
941
942 -- Refill the kids' logon name field
943 fillStudentName()
944
945 end
946
947 -- script of cast member seeUserList
948
949 on mouseUp
950
951   global instanceOfXtra
952
953
954   put "" into field "userList"
955
956
957   -- Set up where to find the users file
958   put the pathName & "users" into whatFile
959
960
961   -- Start up Fileio Xtra
962   set instanceOfXtra = new(xtra "fileio")
963
964
965   -- Set up Fileio to read from users file
966   openFile(instanceOfXtra, whatFile, 1)
967
968
969   -- If file users doesn't exist, create it and set it up for read to avoid error
970
971   if status(instanceOfXtra) <> 1 then
972     createFile(instanceOfXtra, whatFile)
973     openFile(instanceOfXtra, whatFile, 1)
974   end if
975
976
977   -- Read what's currently in the file
978   set whatText = readFile(instanceOfXtra)
979
980
981   -- Put the updated user list into the userList field
982   put whatText into field "userList"
983
984   -- Close Fileio Xtra
985   closeFile(instanceOfXtra)
986

```


Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
987 set instanceOfXtra = 0
988
989 end
990
991
992
993 -- script of cast member DoneAdmin
994
995 on mouseUp
996
997 go to frame "open"
998
999 put "" into field "addName"
1000 put "" into field "addUserGroup"
1001 put "" into field "addPass"
1002 put "" into field "addMailFileLoc"
1003
1004 end
1005 -- msgHandlers scripts
1006 -----
1007 --- openMsgHandler starts the appropriate Message Handling movie.
1008 --- The call must be continued in emh_continue.
1009 --- It is necessary that the global variable emG_mailData is
1010 --- set up. Therefore, we pass it as a parameter to make it
1011 --- clear that the variable is necessary.
1012
1013 on openMsgHandler mimeType, mailData
1014
1015 set movieName = getMessageHandler(mimeType)
1016 go to frame "movie"
1017
1018 -- since all sprites are automatically puppets in Dir 6.0
1019 -- next should not be necessary
1020 -- Take control of the sidebar buttons
1021
1022 puppetSprite 6, TRUE
1023 puppetSprite 7, TRUE
1024 puppetSprite 8, TRUE
1025 puppetSprite 9, TRUE
1026
1027 set mshMovie = window movieName
1028 set the titleVisible of mshMovie to FALSE
1029 set the rect of mshMovie = getMovieRect(mimeType)
1030
1031 open mshMovie
1032 set the name of mshMovie to "childWindow"
1033
1034 tell window "childWindow"
1035 -- next is a hack to get around Macromedia MIAW bug
1036 -- see emh_continue for calls to real handlers
1037 emc_startMeUp()
1038
1039 end tell
1040
1041 -- CONTINUES in emh_continue
1042 end openMsgHandler
1043
1044
```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```

1045 -----
1046 -- getMessageHandler returns filename of movie to handle mimetype.
1047 -- This code makes it easy to make changes in movie filenames
1048 -- and to add new message handling movies.
1049
1050 on getMessageHandler mimetype
1051
1052   case mimetype of
1053     "text": return("text.dir")
1054     "rebus": return("rebus.dir")
1055     "grid": return("grid.dir")
1056     "connect": return("connect.dir")
1057     "puzzle" : return("puzzle.dir")
1058
1059     otherwise:
1060       alert "Invalid mimetype of message."
1061       return("")
1062   end case
1063
1064 end getMessageHandler
1065 -----
1066
1067 on getMovieRect whichMovie
1068
1069   --- the top of green panel
1070   set movieTop = the top of sprite 3
1071   --- the left of green panel
1072   set movieLeft = the left of sprite 3
1073
1074   case whichMovie of
1075     "rebus", "rebus.dir":
1076       set theRect= rect(movieLeft, movieTop, -
1077         movieLeft + 640, movieTop +480)
1078     "text", "text.dir":
1079       set theRect= rect(movieLeft, movieTop, -
1080         the stageRight - 5, the stageBottom -5)
1081     "puzzle", "puzzle.dir":
1082       set theRect= rect(movieLeft, movieTop, -
1083         the stageRight - 5, the stageBottom -5)
1084     "grid", "grid.dir", "connect", "connect.dir":
1085       set theRect= rect(movieLeft, movieTop, -
1086         the stageRight - 5, the stageBottom -5)
1087     "mailbox", "mailbox.dir":
1088       set theRect= rect(movieLeft, movieTop, -
1089         the stageRight - 5, the stageBottom -5)
1090     otherwise:
1091       alert "ERROR: invalid movieName: " & whichMovie
1092       set theRect = rect(0,0,0,0)
1093
1094   end case
1095
1096   return(theRect)
1097
1098 end getMovieRect
1099
1100
1101
1102 -- score script fr_installMenu

```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
1103
1104 on prepareFrame
1105   --first clear away any old menus
1106   installMenu 0
1107   installMenu "main menu"
1108 end
1109
1110 -----
1111 -- password verification and user init
1112
1113
1114
1115 on enterMainEmail username
1116   global emG_userName, emG_userGroup, emG_userGroupList
1117
1118   set emG_userName = username
1119   set emG_userGroup = getProp(emG_userGroupList, emG_userName)
1120
1121   -- ADMINISTRATOR has access to the "Edit Users" button
1122   if emG_userName = "administrator" then
1123     set the visible of sprite 20 = TRUE
1124   end if
1125
1126   go to frame "open"
1127
1128 end enterMainEmail
1129
1130 -----
1131
1132
1133 on checkUserName userName
1134   global emG_registeredUsers
1135
1136   if getone(emG_registeredUsers, userName) then
1137     return(1) -- username is in system
1138
1139   else
1140     alert "User " & userName & "not a KidCode authorized user." & RETURN & "You
1141     cannot login without a valid user name."
1142
1143   end if
1144
1145 end checkUsername
1146
1147 -- more password handling scripts
1148
1149 on checkPassword userName, password
1150   global emG_passwordList
1151
1152   -- if the username is not valid quit this...
1153   if not checkUserName(userName) then return(0)
1154
1155   -- username is valid
1156
1157   -- First part of loop changes capital letters to lower case
1158   -- Second part puts lower case letters into password check
1159   -- This eliminates all spaces and/or unacceptable characters
1160
```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```

1161 set checkPassword = ""
1162 repeat with i = 1 to the number of chars in password
1163
1164     put char(i) of password into capital
1165     put charToNum(capital) into capital
1166
1167     if capital <= 90 and capital >= 65 then
1168         put numToChar(capital + 32) after checkPassword
1169     else if capital >= 97 and capital <= 122 then
1170         put numToChar(capital) after checkPassword
1171     end if
1172
1173 end repeat
1174
1175 -- CHECK PASSWORD
1176
1177 set realPassword = getProp(emG_passwordList, username)
1178
1179 if realpassword = checkPassword then
1180     return(1) --TRUE
1181 else
1182     return(0)
1183 end if
1184
1185
1186 end checkPassword
1187
1188 -----
1189
1190 on clearPassword
1191     global gpw, gpwlen
1192
1193     set gpw = ""
1194     set gpwlen = 0
1195     put "" into field "StudentPass"
1196
1197 end clearPassword
1198
1199 -----
1200
1201 on alertBadPassword
1202
1203     set the loc of sprite 17 to point(231, 350)
1204     beep()
1205
1206 end alertBadPassword
1207
1208 on hideAlert
1209     set the loc of sprite 17 to point(-188, -31)
1210
1211 end hideAlert
1212
1213
1214 -- script of cast member reply
1215
1216 on mouseUp
1217     global emG_userName, emG_msgNumber
1218     global emG_maildata, emG_mode, emG_userGroup

```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
1219
1220 -- abandon current MailData which should be in the inbox.
1221 -- Later, the user may choose to either abandon or send
1222 -- the new replyTo message. That is not a concern.
1223
1224 -- If a mailbox window is open need to get the message
1225 -- and close that window.
1226
1227 tell window "childWindow" to emc_getComponentInfo()
1228 set cInfo = the result
1229 if getComponentProp(cInfo, #ComponentType) = #mailbox then
1230   tell window "childwindow" to mbx_GetMessageNumber()
1231   set emG_msgNumber = the result
1232   if emG_msgNumber <= 0 then
1233     alert "You must select a message."
1234     return() -- abandon the request to reply
1235   end if
1236
1237   tell window "childwindow" to mbx_GetMessage(emG_msgNumber)
1238   set emG_maildata = the result
1239
1240   --- forget window "childwindow" -- done in passMessage
1241
1242   --- Now open the appropriate Message Handler
1243   --- to display the message
1244
1245   emh_PassMessage(emG_maildata, emG_msgNumber)
1246
1247 end if
1248
1249 -- If we got to this point message handler is open.
1250 -- Presumably it has a message displayed. If the message
1251 -- is empty only the message handler knows that and it
1252 -- will need to catch the error and return an error code
1253 -- to msh_replyMessage.
1254
1255 -- The message handling movie's replyMessage handler
1256 -- should swap "to" and "from"
1257 -- fields and make the message editable
1258
1259 -- set mode to author to keep it consistent with msg handler
1260 set emG_mode = #author
1261
1262 set emG_msgNumber = 0 -- this is now a new message
1263
1264 tell window "childWindow"
1265   global emG_userGroup
1266   -- msg handler will swap "to" with "from" and change
1267   -- mode to author
1268   moveToFront window "childWindow"
1269   msh_replyMessage()
1270 end tell
1271
1272 set emG_maildata = the result
1273
1274 -- Toggle the send and reply buttons
1275 setreply -- disable reply and enable send buttons
1276
```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
1277
1278 end
1279
1280
1281 -- script of cast member send
1282
1283 on mouseUp
1284   global emG_maildata, emG_userGroup
1285
1286   -- Could check that the childwindow is a messagehandler
1287   -- but this may not be necessary.
1288
1289   tell window "childWindow"
1290     global emG_userGroup
1291     msh_sendMessage()
1292     set emG_maildata = the result
1293   end tell
1294
1295   if not isValidMessage(emG_maildata) then
1296     alert "ERROR not a valid message."
1297     return(0)      -- abandon attempt to send
1298   end if
1299
1300   --- otherwise continue to send message
1301
1302   -- NEED TO FIX THIS SO THAT MESSAGE STATUS DOES NOT
1303   -- BECOME "#sent" if it fails to be saved to both
1304   -- mail files
1305
1306   messageHandler(#sent) -- for now this uses global emG_maildata.
1307
1308   -- tell window "childWindow" to msh_clearMessage()
1309
1310 end
1311
1312 -- script of cast member print
1313
1314 on mouseUp
1315
1316   tell window "childwindow" to emc_getComponentInfo()
1317   set cInfo = the result
1318   set cType = getComponentProp(cInfo, #ComponentType)
1319
1320   if cType = #mailbox then
1321     -- need to pass the message to its message handling
1322     -- component for printing. Ideally this can be done
1323     -- without opening a window and laying out the message.
1324
1325     alert "I can't do that right now. Open the message and then print."
1326
1327   else if cType = #msgHandler then
1328
1329     tell window "childwindow"
1330       msh_PrintMessage()
1331     end tell
1332
1333   else alert "ERROR invalid componenttype."
1334
```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
1335 end
1336
1337 -----
1338 -- script of cast member Quit
1339
1340 on mouseUp
1341     handleQuit()
1342
1343 end
1344
1345 -----
1346
1347 on handleQuit
1348     initializeUser()
1349     clearPassword()
1350     go to frame 2
1351
1352     -- make sure the editUsers button is invisible
1353     set the visible of sprite 20 = FALSE
1354
1355 end handleQuit
1356
1357 -- script of cast member trash
1358
1359 --- Email Main now handles all aspects of trashing a
1360 --- message by writing the mail files. The components
1361 --- are instructed to update their state by clearing the
1362 --- message (if the component is a message handler) or
1363 --- redrawing the message list (if the component is a
1364 --- mailbox.)
1365
1366 --- Should add a confirmation dialog with the user
1367
1368 on mouseUp
1369     global emG_msgNumber -- number of the current message
1370
1371     tell window "childwindow" to emc_getComponentInfo()
1372     set cInfo = the result
1373     set cType = getComponentProp(cInfo, #ComponentType)
1374
1375     if cType = #mailbox then
1376         -- need to determine which message(s) are currently
1377         -- selected and instruct the mailbox to update its
1378         -- display
1379
1380         -- temporary implementation of mbx_trashMessages does
1381         -- not handle multiple messages as a result the
1382         -- arguments are ignore...
1383
1384         tell window "childwindow" to mbx_trashMessages([])
1385
1386         -- the following lines will be necessary when
1387         -- mbx_trashMessages is properly implemented. For
1388         -- now, the temporary implementation trashes the
1389         -- message itself.
```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
1393 -- set messageNumbers = the result
1394 -- delete each message in the list of messageNumbers
1395
1396
1397 else if cType = #msgHandler then
1398
1399 -- rewrite the message into the mailfile
1400 messageHandler(#trash)
1401
1402 tell window "childwindow" to msh_clearMessage()
1403
1404 else alert "ERROR invalid componentype."
1405
1406
1407 end
1408
1409
1410 -- script of cast member text
1411
1412 on mouseUp
1413   global emG_msgNumber
1414   global emG_maildata, emG_mode
1415
1416   -- START A NEW MESSAGE
1417
1418   set emG_msgNumber = 0
1419   set emG_mode = #author
1420   set emG_maildata = createMailData(emG_userName, "text")
1421
1422   openMsgHandler("text", emG_mailData)
1423
1424   disableReply()
1425
1426 end
1427
1428 -----
1429 -- script of cast member Rebus
1430
1431 on mouseUp
1432   global emG_msgNumber
1433   global emG_maildata, emG_mode
1434
1435   -- START A NEW MESSAGE
1436
1437   set emG_msgNumber = 0
1438   set emG_mode = #author
1439   set emG_maildata = createMailData(emG_userName, "rebus")
1440
1441   openMsgHandler("rebus", emG_mailData)
1442
1443   disableReply()
1444
1445 end
1446
1447
1448 --- script of cast member grid
1449
1450 on mouseUp
```


Appendix A: "KidCode® Lingo Client/Server Email Main Scripts

```
1451 global emG_msgNumber
1452 global emG_maildata, emG_mode
1453
1454 -- START A NEW MESSAGE
1455
1456 set emG_msgNumber = 0
1457 set emG_mode = #author
1458 set emG_maildata = createMailData(emG_userName, "grid")
1459
1460 openMsgHandler("grid", emG_mailData)
1461
1462 disableReply()
1463
1464 end
1465
1466
1467 --- script of cast member puzzle
1468
1469 on mouseUp
1470 global emG_msgNumber
1471 global emG_maildata, emG_mode
1472
1473 -- START A NEW MESSAGE
1474
1475 set emG_msgNumber = 0
1476 set emG_mode = #author
1477 set emG_maildata = createMailData(emG_userName, "puzzle")
1478
1479 openMsgHandler("puzzle", emG_mailData)
1480
1481 disableReply()
1482
1483 end
1484
1485
1486 --- script of cast member connect
1487
1488 on mouseUp
1489 global emG_msgNumber
1490 global emG_maildata, emG_mode
1491
1492 -- START A NEW MESSAGE
1493
1494 set emG_msgNumber = 0
1495 set emG_mode = #author
1496 set emG_maildata = createMailData(emG_userName, "connect")
1497
1498 openMsgHandler("connect", emG_mailData)
1499
1500 disableReply()
1501
1502 end
1503
1504
1505 on GetComponentProp infoList, prop
1506
1507 --- need to add error checking code
1508
```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
1509 case prop of
1510   #componentName: return(getAt(infolist, 1))
1511   #componentID: return(getAt(infolist, 2))
1512   #componentType: return(getAt(infolist, 3))
1513   #componentMIMEtype: return(getAt(infolist, 4))
1514
1515   otherwise: alert "ERROR no component property."
1516 end case
1517
1518 end GetComponentProp
1519
1520 -- script of cast member savebox
1521
1522 on mouseUp
1523   openMbx("savebox")
1524
1525
1526 end
1527
1528
1529 -- script of cast member inbox
1530
1531 on mouseUp
1532   openMbx("inbox")
1533
1534
1535 end
1536
1537
1538
1539 -- script of cast member outbox
1540
1541 on mouseUp
1542   openMbx("outbox")
1543
1544
1545 end
1546
1547
1548 --- Users File functions
1549
1550 -- returns a string of all users data from the users file.
1551
1552 -- THIS FUNCTION NEEDS TO CHECK THAT DATA IS VALID
1553
1554 on readUsersFile
1555
1556   -- Set up where to find the users file
1557   put the pathName & "users" into whatFile
1558
1559   -- Start up Fileio Xtra
1560   set instanceOfXtra = new(xtra "fileio")
1561
1562   -- Set up Fileio to read from users file
1563   openFile(instanceOfXtra, whatFile, 1)
1564
1565
1566   -- If file users doesn't exist, create it
```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```

1567
1568 if status(instanceOfXtra) <> 0 then
1569     createFile(instanceOfXtra, whatFile)
1570     openFile(instanceOfXtra, whatFile, 1)
1571 end if
1572
1573
1574 -- Read what's currently in the file
1575 set whatText = readFile(instanceOfXtra)
1576
1577
1578 -- if no users are defined, assume administrator as default user
1579 -- Administrator info is not written into the user's file until at
1580 -- least one user is defined. This occurs in AddUsers functions.
1581
1582 if whatText = "" then
1583     -- for now, assume admin has mail file in each
1584     -- location where kidcode is installed
1585     put "administrator,kidcode,0," & the pathName & "administrator" & RETURN into
1586     whatText
1587 end if
1588
1589 -- Close Fileio Xtra
1590 closeFile(instanceOfXtra)
1591 set instanceOfXtra = 0
1592
1593 return(whatText) -- string read from users file
1594
1595 end readUsersFile
1596
1597
1598 -----
1599 -- more users file scripts
1600
1601 on writeUsersFile
1602     global emG_registeredUsers, emG_passwordList, emG_userGroupList,
1603     emG_mailFileList
1604
1605     -- Set up where to find the users file
1606     put the pathName & "users" into whatFile
1607
1608     -- Start up Fileio Xtra
1609     set instanceOfXtra = new(xtra "fileio")
1610
1611     -- Set up Fileio to read and write from/to users file
1612     openFile(instanceOfXtra, whatFile, 0)
1613
1614     -- If file users doesn't exist, create it and set it up for read/write
1615
1616     if status(instanceOfXtra) <> 0 then
1617         createFile(instanceOfXtra, whatFile)
1618         openFile(instanceOfXtra, whatFile, 0)
1619     end if
1620
1621     -- Put the cursor at the beginning of the users file
1622     setPosition(instanceOfXtra, 0)
1623
1624     --- put together string of usersData

```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
1625 set whatText = ""
1626 repeat with uname in emG_registeredUsers
1627
1628     set pw = getProp(emG_passwordList, uname)
1629     set ugroup = getProp(emG_userGroupList, uname)
1630     set mfile = getProp(emG_mailFileList, uname)
1631     set whatText = whatText & uname & "," & pw & "," & ugroup & "," & mfile &
1632 RETURN
1633
1634 end repeat
1635
1636 -- Overwrite users file with updated list
1637 writeString(instanceOfXtra, whatText)
1638
1639 -- Close Fileio Xtra
1640
1641 closeFile(instanceOfXtra)
1642
1643 set instanceOfXtra = 0
1644 return(1)
1645
1646 end writeUsersFile
1647
1648 -----
1649 --- these next functions are created to do file checking
1650 --- however they appear to suffer from severe crash problems
1651 --- these problems will also effect mail file creation if
1652 --- path names are invalid...we need to fix this
1653
1654 on pathp pathname
1655
1656     set instanceOfXtra = new(xtra "fileio")
1657     openFile(instanceOfXtra, pathname, 1)
1658     set theval = status(instanceOfXtra)
1659
1660     case theval of
1661         0 :
1662             closeFile(instanceOfXtra)
1663             set instanceOfXtra = 0
1664             return(1)
1665
1666         -36: -- I/O Error...likely to cause system crash
1667             alert "System has become unstable. " & RETURN & "Please save your work."
1668             -- next call to fileio xtra may crash system
1669             set instanceOfXtra = 0
1670             return(0)
1671
1672     otherwise :
1673         alert " " & error(instanceOfXtra, theval)
1674         closeFile(instanceOfXtra)
1675         set instanceOfXtra = 0
1676         return(0)
1677
1678     end case
1679
1680 end pathExists
1681
1682
```

Appendix A: KidCode® Lingo Client/Server Email Main Scripts

```
1683 on foldertest
1684   getNthFileNameInFolder("C:\windows", 1)
1685 end foldertest
```

KidCode® Application Programming Interface (API)

This API defines the data and function calls that are used for communication between the KidCode Main Email program and installable components. Each installable component can be one of two types:

- mailbox browser/editor component
- message authoring/display component

KidCode Main Email application may communicate with another mail server such as an SMTP compliant server to retrieve and store email messages. Alternatively, the Email Main program may include code for many of the functions normally associated with a mail server program. Whether in conjunction with a mail server, or on its own, the Email Main program handles all functions associated with sending and receiving email messages. This includes reading and writing mailbox files to/from permanent storage or other mail servers on a network (e.g. using POP3), finding and verifying network addresses, and sending mail messages to other servers on a network.

The Main Email Program also provides a GUI that provides interaction with a user for those functions that are directly associated with storage and transfer of electronic mail messages and mailboxes. In particular, the Main Email program includes buttons and/or menu items that allow a user to:

- **Send** (a message),
- **Reply** (to a message),
- **Open** (a message or a mailbox),
- **Delete/Trash** (messages or mailboxes),
- **Save** (a message to an alternative mailbox)
- **Print** (a message)

The Main Email Program also handles all data bundling and unbundling that may be required to transform the message data used by a message authoring component into a fully MIME compliant message type. This way each message authoring component can handle data in a format most convenient to it and all MIME parsing and details associated with protocol compliance can be centralized in the Main Email application. The only requirement for the message data passed between a message authoring component and the Main Email Program is that the message body data be formatted either as an ASCII string or in a binhex format.

The KidCode Main Email program communicates with installable components in order to execute the commands defined above.

Mailbox browser/editor components

Mailbox components are used to display, edit, and browse mailboxes. Different kinds of users and different types of messaging applications (e.g. fax, traditional email, internet voice) may require very different displays and functionality from a mailbox viewer/editor. Installable mailbox components make it possible to upgrade, select from multiple viewing formats, utilize different mailbox viewer/editors for different users, and in general increase the range of functionality that can be achieved within one basic messaging application program.

Message authoring/display components

Message handler components make it possible to handle an unlimited number of message types. Each message handler component is designed to deal with a specific MIME type of message. The MIME data standard has been designed so that application developers can define new MIME types as needed by labeling these with the "/application-x" prefix. A message handler component can be any program that defines a message MIME type of

Appendix B: KidCod @ API

data that it handles and that implements the callback entry points described in this document. These functions allow the Main Email application to obtain information about the message handler and allows the message handler to respond to standard mail commands such as Send or Reply, that have been issued by a user through the Main Email interface. Example message handler components included in the KidCode application are an ordinary ascii text message handler, a game called Rebus that allows users to create and respond to graphical rebus messages, an a sample mathematics workbook that allows students and a teacher to send workbook problems to one another.

Global variable naming conventions:

Each movie should name its global variables with a prefix that identifies the movie and a capital "G" for "global". We will keep track of each movie's prefix. For now we have the following identifying prefixes:

component prefix	component	global variable prefix
em_	main movie	emG_
tm_	text movie	tmG_
rm_	rebus movie	rmG_
cm_	connect movie	cmG_
tgm_	text grid movie	tgmG_
pm_	puzzle movie	pmG_
mbx_	mailbox movie	mbxG_

Main Movie Public Data Types

em_ComponentType symbol = #mailbox or #msgHandler

em_UserName string

```
em_UserData struct (
    str    UserName
    str    FullName
    str    ReturnAddress
    em_AddressBook AddressBook
    em_MailboxList Mailboxes
    str    SMTPHost
    str    POP3Host
    str    Password
)
```

em_MailboxName string

```
em_Mailbox struct (
    em_mailboxName boxName
    list of emMailData
)
```

em_RegisteredUsers list of em_UserName

Appendix B: KidCode® API

```

em_MailData struct (
    em_Address      To
    em_Address      From
    str             Re
    str             Data
    str             MimeType
    list            MsgBody
)

em_MessageNumber int

em_Mode symbol = #author or #display

em_ComponentInfo struct (
    str             ComponentName
    int             ComponentID
    em_ComponentType ComponentType
    str             ComponentMIMEType ; nil if mailbox
)

```

Email Main API Functions

These functions are called by the installable components to access services provided in the KidCode Main Email program.

```

/
*****
****/

```

/* emh_getUserMailbox

Return a mailbox data structure for the current user and mailbox name. This function is normally called by a mailbox handling component. Mailbox handling components may use temporary files to hold mailbox contents but they should never access the users mailbox files. All access to these files must be obtained through the Main Email program.

*/

```

em_Mailbox emh_getUserMailbox (
    em_MailBoxName
)

```

```

/
*****
****/

```

/* emh_getUserData

Return a data structure with user information. The KidCode Main Email program maintains all user information and handles user administration functions. The Main program also communication with external Mail servers which may contain other user information not part of the KidCode API.

*/

```

em_UserData emh_getUserData (
    em_UserName,

```


Appendix B: KidCode® API

```

)

/
*****
****/
/* emh_continue
Used by installable components to explicitly pass control back to the Main Email
program. This function is necessary for the Director/Lingo implementation.
*/

void emh_continue (
    em_ComponentType
)

/
*****
****/
/* emh_killComponent
Used by an installable component to inform the Main Email program that it is preparing
to terminate. This allows the Main program to free any memory and/or data structures that
have been allocated to the component.
*/

void emh_killComponent (
)

/
*****
****/
/* emh_passMessage
Used primarily by mailbox components to pass a message to the Main program so that it
can be displayed by the appropriate message handling component. Email main takes the
message argument (em_MailData, looks up the Mimetype of the message, and invokes the
appropriate message handler to display the message.
*/

void emh_passMessage (
    em_MailData,
    em_MessageNumber
)

/
*****
****/
/* emh_getMessage
Returns the message (em_MailData) with Number MessageNumber from the
MailboxName of the current user. Can be used by installable components to retrieve
specific messages from the user's mailboxes.

If this is called with the messageNumber set to 0, email main assume the typeOrBoxName
argument is a mimetype and returns a new message structure. Message handling
components should call emh_getMessage with the number set to 0 and the mimetype

```

Appendix B: KidCode® API

whenever a new message is started. Normally this should be done whenever an active message is trashed.

*/

```
em_MailData emh_getMessage (
    em_MessageNumber
    str      typeOrBoxName
)
```

/

****/

/* emh_getRegisteredUsers

Returns a list of usernames for the users that are registered with the KidCode system, i.e. that have been added as users by the User Administration part of the Main Email Program. This is the same list of users that appear in the logon listbox when the program is started up. It may be used by installable components to create listboxes for filling address fields in messages or for checking on whether a particular address is external to the system.

*/

```
em_RegisteredUsers emh_getRegisteredUsers (
)
```

/

****/

/* emh_sendMessage

Email Main sends the message argument (em_MailData) by either forwarding to an external mail server or, if it is a registered KidCode user, writing the message to the user's incoming mail mailbox.

*/

```
void emh_sendMessage (
    em_MailData
)
```

/

****/

/* emh_saveMessage

Email Main saves the message argument (em_MailData) for the currently logged on user by writing the message to the user's "notes in progress" mail mailbox.

*/

```
void emh_saveMessage (
    em_MailData
)
```

/

Appendix B: KidCode® API

```
****/
/* emh_disableButton
It is recommended that this function be used carefully. Normally Email Main controls
the state of all the buttons available to users to access message handling of the main
program (i.e. buttons in the purple left hand panel). This function can be used to request
that Email Main disable the button specified by the argument, ButtonName. If the button
is disabled - whether it was already disabled or is disabled as a result of the function call -
the function will return TRUE, otherwise it will return FALSE. The calling component
should check on whether the function call succeeded and proceed accordingly.
*/
```

```
em_ReturnValue emh_disableButton (
    str          ButtonName
)
```

```
/
*****
****/
```

```
/* emh_enableButton
It is recommended that this function be used carefully. Normally Email Main controls
the state of all the buttons available to users to access message handling of the main
program (i.e. buttons in the purple left hand panel). This function can be used to request
that Email Main enable the button specified by the argument, ButtonName. If the button
is enabled - whether it was already disabled or is disabled as a result of the function call -
the function will return TRUE, otherwise it will return FALSE. The calling component
should check on whether the function call succeeded and proceed accordingly.
*/
```

```
em_ReturnValue emh_enableButton (
    str          ButtonName
)
```

API Functions Required Implementation of all Component Types

```
/
*****
****/
```

```
/* emc_startMeUp
Used by Email Main to tell an installable component to start. This function will execute
prior to initialization of the component's data structures. Which should only be initialized
after the component receives the emc_initWindow call from Email Main.
This function is necessary for the Director/Lingo implementation.
*/
```

```
em_ReturnValue emc_startMeUp (
)
```

```
/
*****
****/
```

```
/* emc_initWindow
Used by Email Main to tell an installable component to initialize it's data structures and
```

Appendix B: KidCode® API

prepare its graphical display. The component is passed the username of the current user. If it requires additional user information in order to initialize, it can call `emh_getUserInfo` within its implementation of this function.

```
*/
```

```
em_ReturnValue emc_initWindow (
    em_UserName
)
```

```
/
```

```
*****
****/
```

```
/* emc_closeWindow
```

Used by Email Main to tell an installable component to free all memory that it has used, close its window, and shut down.

```
*/
```

```
em_ReturnValue emc_closeWindow (
)
```

```
/
```

```
*****
****/
```

```
/* emc_getComponentInfo
```

Used by Email Main to get required information such as `componentName`, `componentID`, etc. from the installable component.

```
*/
```

```
em_ComponentInfo emc_getComponentInfo (
)
```

API Functions required of a Mailbox Handler Component

```
/
```

```
*****
****/
```

```
/* mbx_getMessageNumbers
```

Used by Email Main to get the message number of the currently selected message in the mailbox browser. If no message is selected, the list should be empty.

```
*/
```

```
list of int mbx_getMessageNumbers (
)
```

```
/
```

```
*****
****/
```

```
/* mbx_getMessage
```

Used by Email Main to get the message data structure of the message with `em_MessageNumber` from the mailbox currently displayed in the mailbox browser. If the function fails, e.g. if there is no message with the given message number, the function

Appendix B: KidCode® API

returns an empty list.

*/

```
em_MailData mbx_getMessage (
    em_MessageNumber
)
```

/

*****/

/* mbx_trashMessages

Used by Email Main to tell the mailbox component to update it's display and it's data structures to delete messages with messageNumbers in the argument list. If the function fails, e.g. if one of the message numbers is invalid, the function returns FALSE, otherwise it returns TRUE. This function should be implemented so that it does not perform partial deletes, i.e. either it succeeds in deleting all of the messages in the list or it should not delete any message.

*/

```
em_ReturnValue mbx_trashMessages (
    list of em_MessageNumber
)
```

/

*****/

/* mbx_openMailbox

Used by Email Main to tell the mailbox component to display the mailbox passed in the argument.

*/

```
em_ReturnValue mbx_openMailbox (
    em_Mailbox
)
```

Functions required of a Message Handler Component

/

*****/

/* msh_sendMessage

Used by Email Main to tell a message handling component to pass back a fully completed message data structure so that it can be sent to the recipient specified in the message's address field. The message handling component should update it's display as appropriate for a message that has been sent. It should also change it's state to #display mode because a message that has already been sent should not be editable. If the function fails, e.g. if a fully completed message cannot be constructed (for example, if the user has not specified a message recipient), the function returns an empty list.

The message handling component will normally control all dialogs with a user that pertain to the message under construction. For example to alert the user to the fact that a message recipient is required. However, if the message handling component fails to pass back a properly formatted, completed message data structure, (or an empty list

Appendix B: KidCod ® API

acknowledging failure) Email Main will detect the error and alert the user about the field or fields that have not been specified.

*/

```
em_MailData msh_sendMessage (
)
```

```
/
*****
****/
```

/* msh_openMessage
Used by Email Main to pass a message data structure to a message handling component so that it can be displayed. The message handling component should display the message in the specified mode - either #author or #display. If the em_Mode argument is #display the message should not be editable. Otherwise the message should be opened so that it can be edited.

If the function fails, e.g. if an error is detected in the message body, the message handler returns FALSE, otherwise the message handler returns TRUE.

*/

```
em_ReturnValue msh_openMessage (
    em_MailData
    em_Mode
)
```

```
/
*****
****/
```

/* msh_replyMessage
Used by Email Main to inform a message handling component to display the currently active message for editing as a reply. In order to reply the message handling component will generally create a new message with the mode set to #author. The new message body may contain material from the original message that is being replied to. In addition, message handling components that handle different player roles may enable or disable various role specific tools at this time. For example, the Rebus message handler will change the RebusState of the new message and enable guessboxes as appropriate.

If the function fails, e.g. if an error is detected in the message body, the message handler returns FALSE, otherwise the message handler returns TRUE.

*/

```
em_ReturnValue msh_replyMessage (
)
```

```
/
*****
****/
```

/* msh_clearMessage
Used by Email Main to inform a message handling component that the current message should be cleared from the display and from the message handling component's data structures. This function is used, for example, when the user indicates they want to trash the current message by clicking on the "trash" button in the Email Main purple panel.

Appendix B: KidCode® API

If the function fails, the message handler returns FALSE. Otherwise the message handler returns TRUE.

*/

```
em_ReturnValue msh_clearMessage (
)
```

/

****/

/* msh_printMessage

Used by Email Main to inform a message handling component that a message should be printed. This function is used, for example, when the user indicates they want to print the current message by clicking on the "print" button in the Email Main purple panel. When the argument, em_mailData, is an empty list, the message handler component should print the currently active message. Otherwise the message handler component should print the message argument. Normally, if the message handler component has been fully initialized and is displayed in a window, Email Main will call this function with an empty list for an argument.

The function may also be used by the Main Email program to have a message handler print a message even though the message handler component has not been fully initialized and displayed in a window. For example, this will occur if an active mailbox component receives a print request from Email Main for a message that has been selected in the mailbox browser. In this case, Email Main will send a request to the appropriate message handler component to print the message without fully starting it up and initializing its window. Therefore the message handler should implement the msh_printMessage function so that the following sequence of function calls succeeds - emc_startMeUp, msh_printMessage(message).

If the function fails, the message handler returns FALSE. Otherwise the message handler returns TRUE.

*/

```
em_ReturnValue msh_printMessage (
    em_MailData
)
```

Claims:

1. An electronic mail client, comprising:
 - a) a plurality of authoring components, a first of said plurality of authoring components for creating a representation of a document including other than text;
 - b) encoding means for automatically encoding said representations created with said authoring components into an Internet-compatible email message; and
 - c) decoding means for automatically decoding said representations encoded by said encoding means.
2. An electronic mail client according to claim 1, wherein:
said plurality of authoring components include at least one installable component.
3. An electronic mail client according to claim 1, wherein:
said plurality of authoring components includes at least one component selected from the group consisting of a game component, a spreadsheet component, and a graphic editor component.
4. An electronic mail client according to claim 1, wherein:
said plurality of authoring components includes at least one component selected from the group consisting of a database component, a presentation component, and a puzzle component.
5. An electronic mail client according to claim 1, wherein:
said encoding means includes MIME-compatible encoding means.
6. An electronic mail client according to claim 1, wherein:
said encoding means includes means for creating a MIME file and means for creating a multipart MIME message,
each of said authoring component cooperating with said encoding means such that a creation of said MIME file and said multipart MIME message is transparent to a user.
7. An electronic mail client according to claim 6, wherein:
said decoding means includes means for concatenating a multipart MIME message and means for decoding a MIME file;
each of said authoring component cooperating with said decoding means such that a concatenation of said multipart MIME message and said decoding of MIME files is transparent to the user.

8. An electronic mail client according to claim 1, further comprising:
 - d) a plurality of installable mailbox/browser components, each of said mailbox/browser components displaying different types of documents in a user's mailbox.
9. An electronic mail client according to claim 1, further comprising:
 - d) a plurality of installable mailbox/browser components, each of said mailbox/browser components displaying mailbox contents in a different style.
10. An electronic mail client according to claim 1, wherein:

said encoding means and said decoding means communicate bidirectionally with said authoring components.
11. An electronic mail client according to claim 1, wherein:

at least one of said authoring components includes means for recognizing whether a user is an author or a reader and for responding differently to authors and readers.
12. An electronic mail client according to claim 1, wherein:

at least one of said authoring components includes means for allowing a user to create a read-only document.
13. An electronic mail client for a student and a teacher, comprising:
 - a) a plurality of authoring components, a first of said plurality of authoring components for creating a representation of a text document and a second of said plurality of authoring components for creating a representation of a document including other than text;
 - b) encoding means for automatically encoding representations created with said authoring components into an email message; and
 - c) decoding means for automatically decoding said representations encoded with said encoding means, wherein

at least one of said authoring components includes means for determining whether the user is the student or the teacher.
14. An electronic mail client according to claim 13, wherein:

said plurality of authoring components include at least one installable component.

15. An electronic mail client according to claim 13, wherein:
said plurality of authoring components includes at least one component selected from the group consisting of a game component, a workbook component, and a graphic editor component.
16. An electronic mail client according to claim 13, wherein:
said plurality of authoring components includes at least one component selected from the group consisting of a database component, a presentation component, and a puzzle component.
17. An electronic mail client according to claim 13, wherein:
said encoding means includes MIME-compatible encoding means.
18. An electronic mail client according to claim 13, wherein:
said encoding means includes means for creating a MIME file and means for creating a multipart MIME message,
each of said authoring components cooperating with said encoding means such that a creation of said MIME file and said multipart MIME message is transparent to the student and the teacher.
19. An electronic mail client according to claim 18, wherein:
said decoding means includes means for concatenating a multipart MIME message and means for decoding a MIME file,
each of said authoring component cooperating with said decoding means such that a concatenation of said multipart MIME message and said decoding of MIME files is transparent to a user.
20. A method of authoring a document and sending it by electronic mail, said method comprising:
a) providing a document-authoring component which authors a document other than a plain-text document;
b) providing a document-encoding component which encodes the document as Internet-compatible email;
c) linking the document-authoring component with the document-encoding component such that documents generated under said document-authoring component are automatically encoded as Internet-compatible email.

21. A method according to claim 20, wherein:
said step of providing a document-authoring component includes providing a plurality of document-authoring components, and
said step of linking includes linking each of said document-authoring components with the document-encoding component.
22. A method according to claim 20, further comprising:
d) providing a document-decoding component which decodes a received document encoded as Internet-compatible email;
c) linking the document-authoring component with the document-decoding component such that documents are automatically decoded.
23. A method according to claim 20, wherein:
the document-encoding component includes means for creating a MIME file and means for creating a multipart MIME message.
24. A method according to claim 22, wherein:
the document-decoding component includes means for concatenating a multipart MIME message and means for decoding a MIME file.

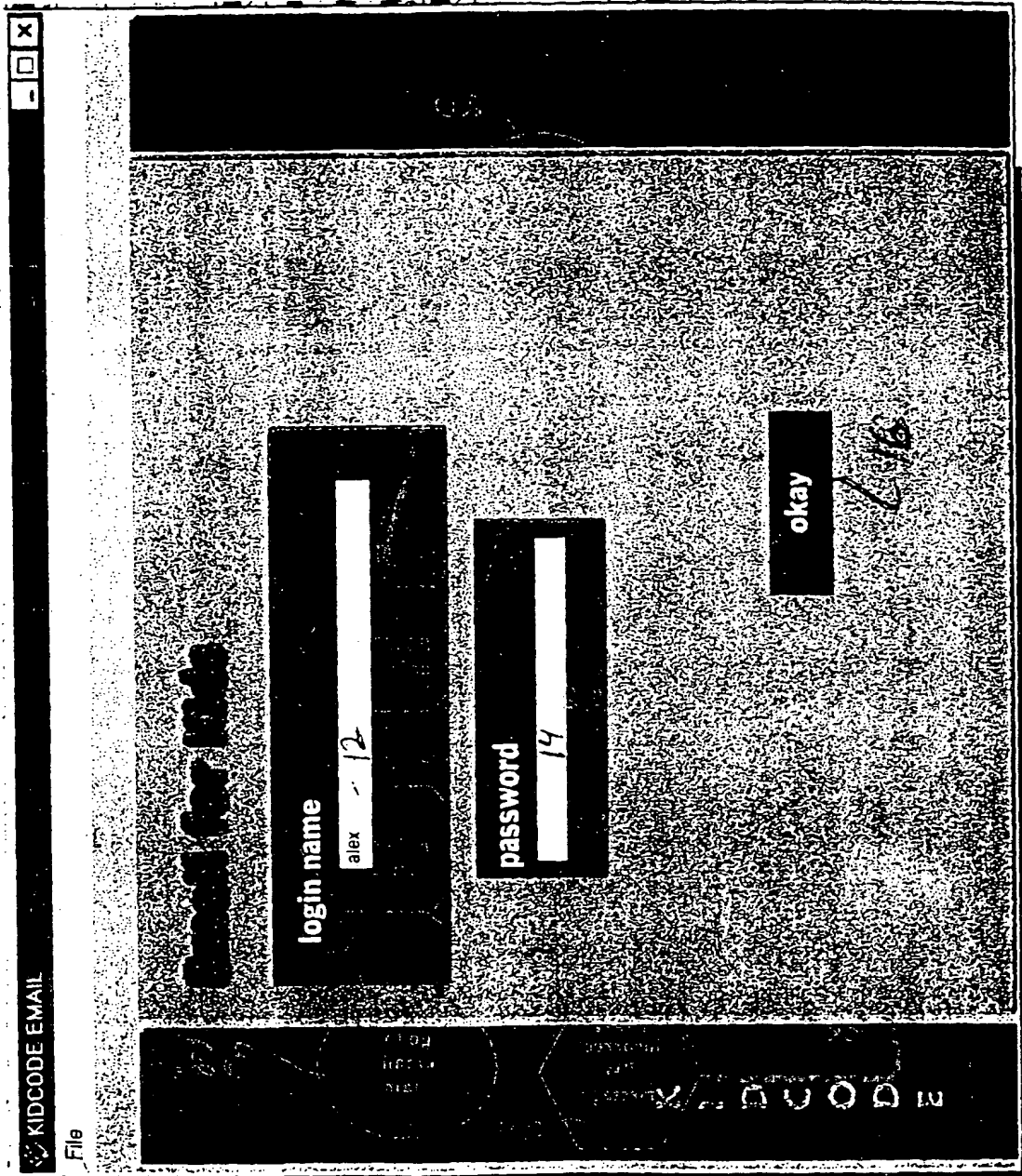
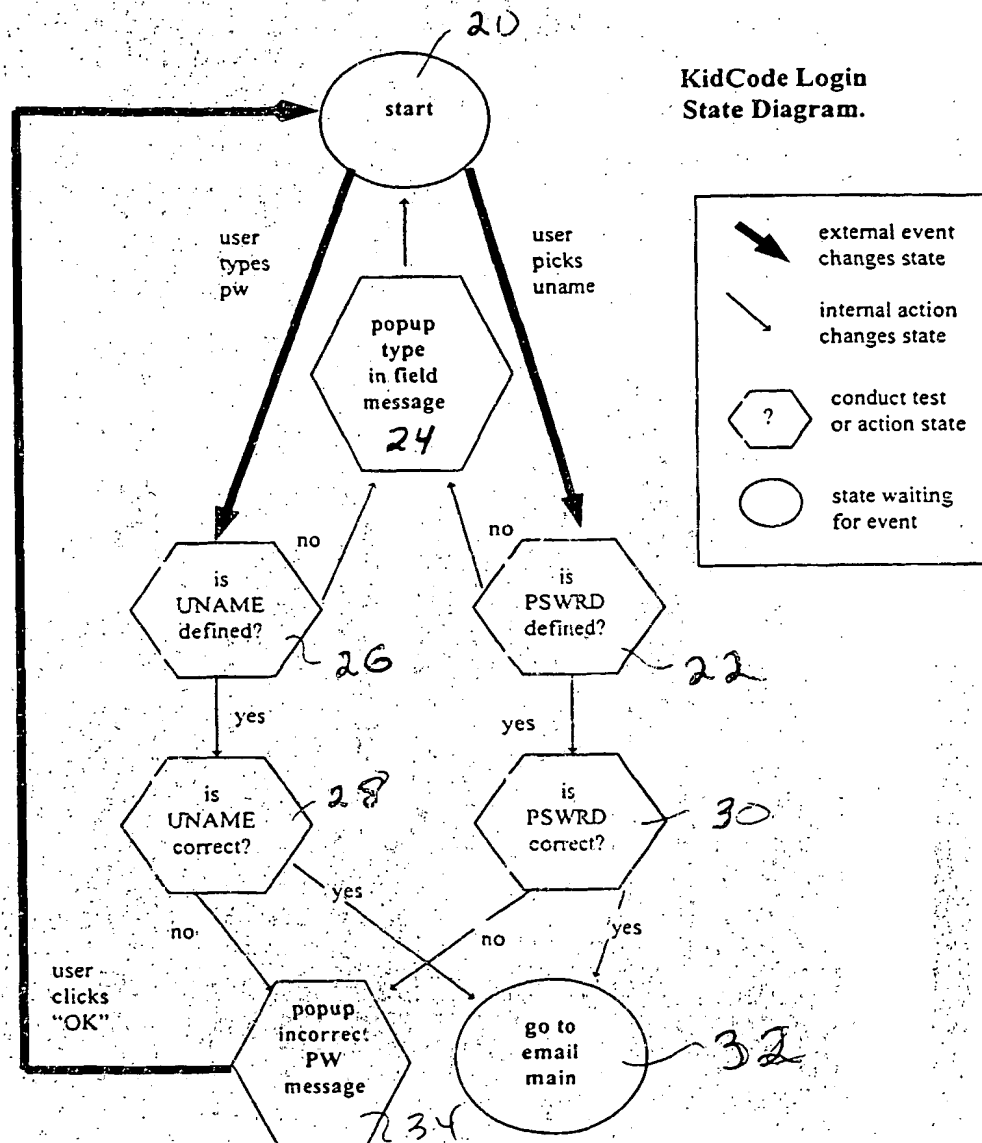
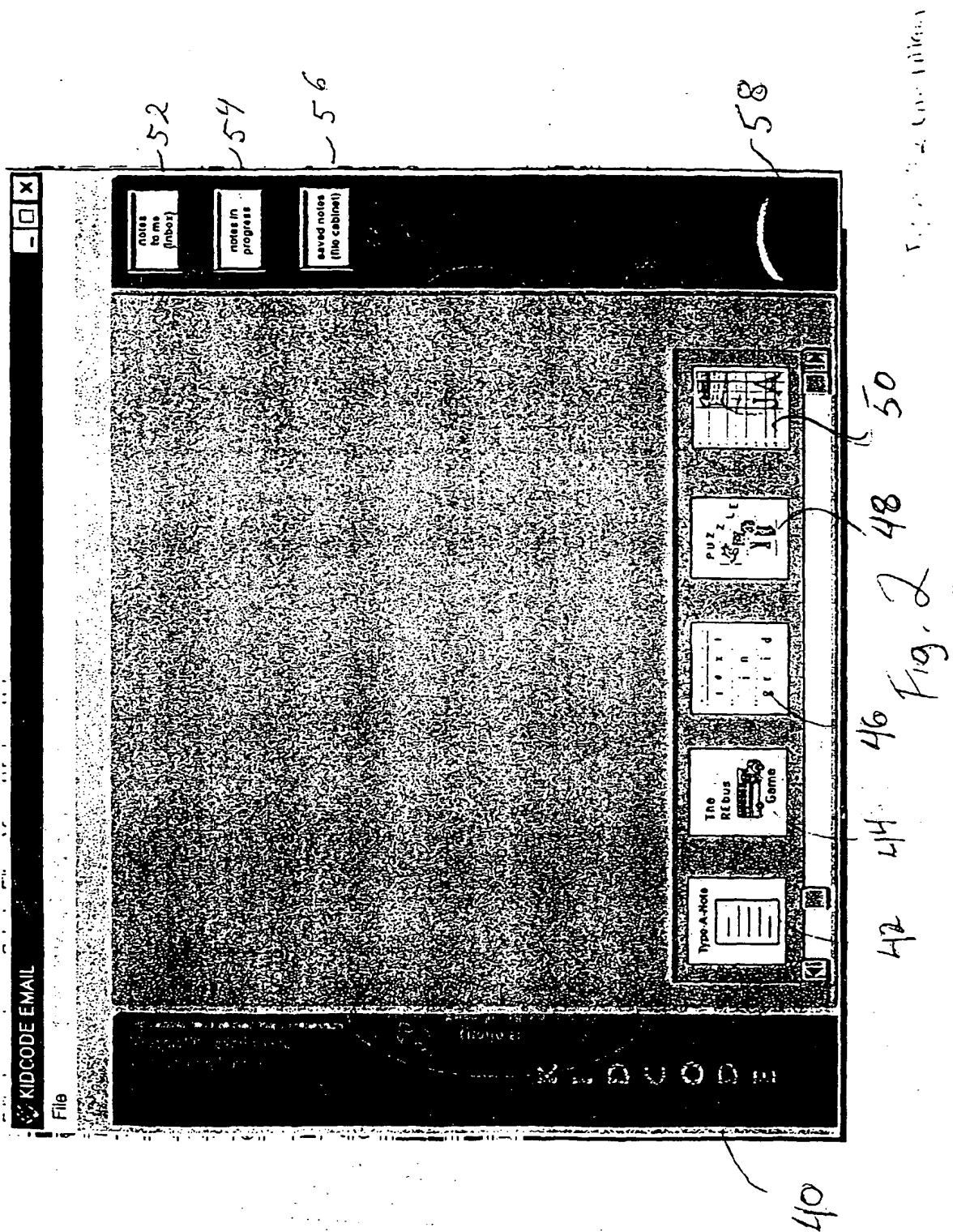


Fig. 1

Fig. 1a



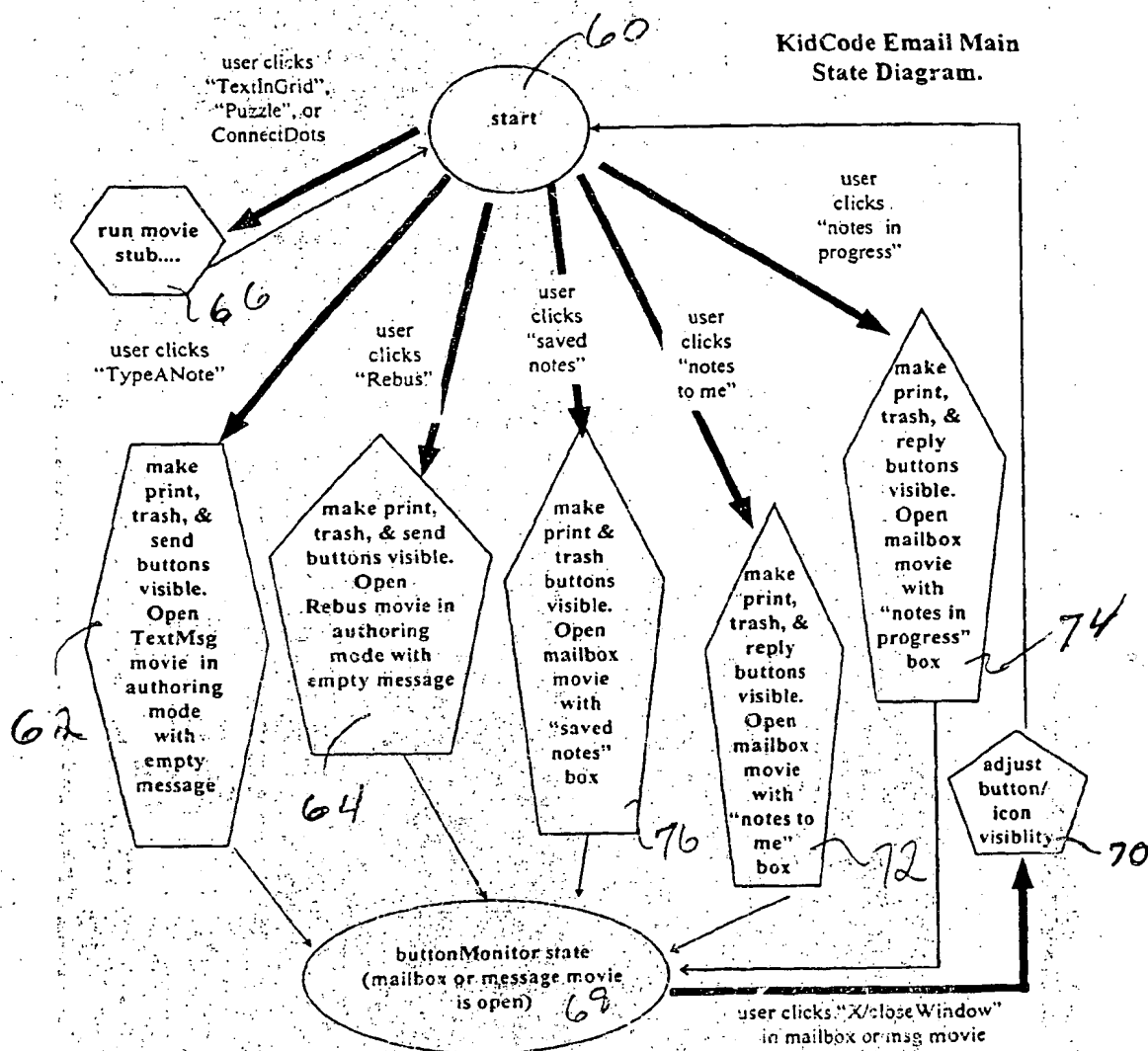
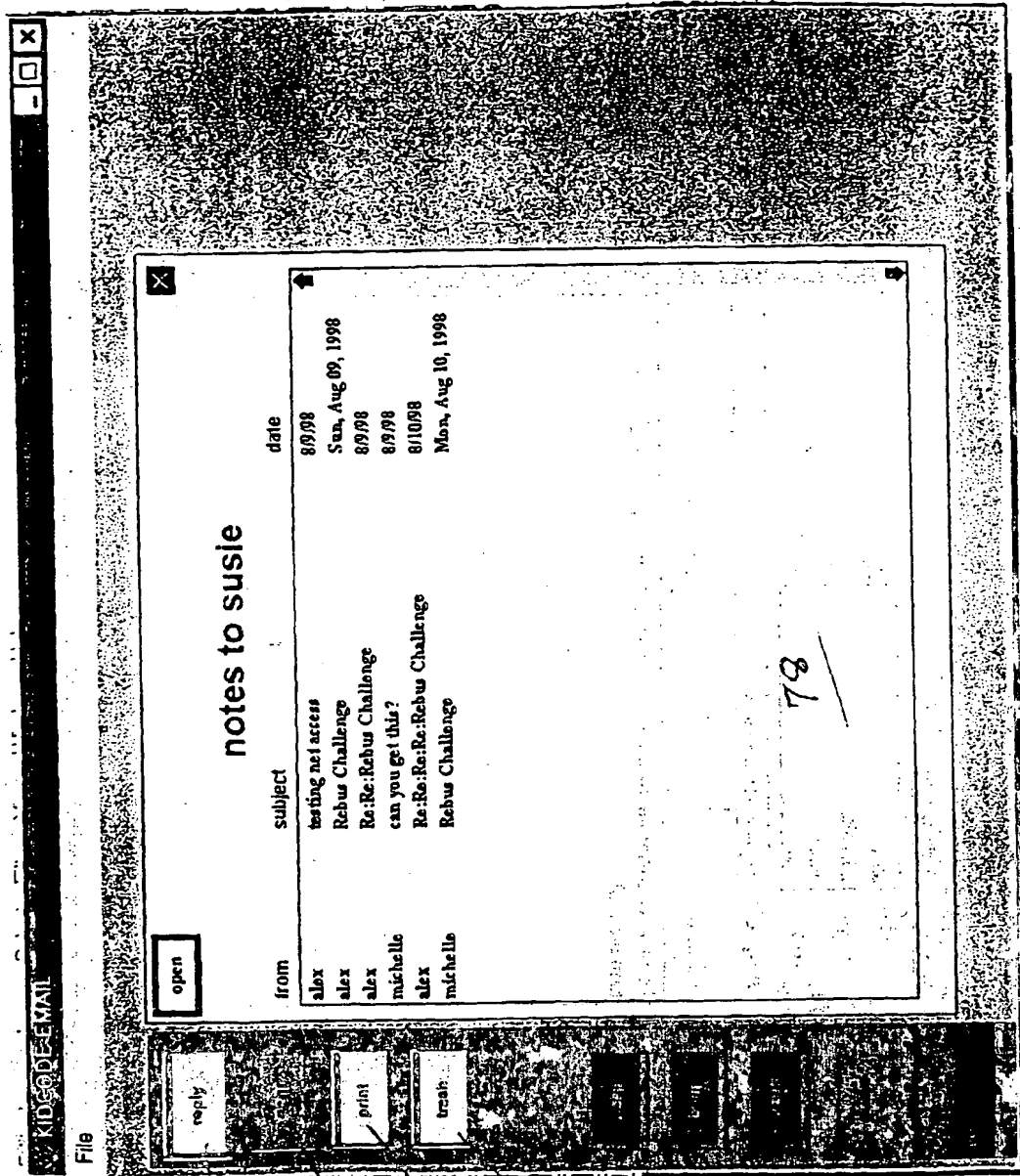


Fig. 2a



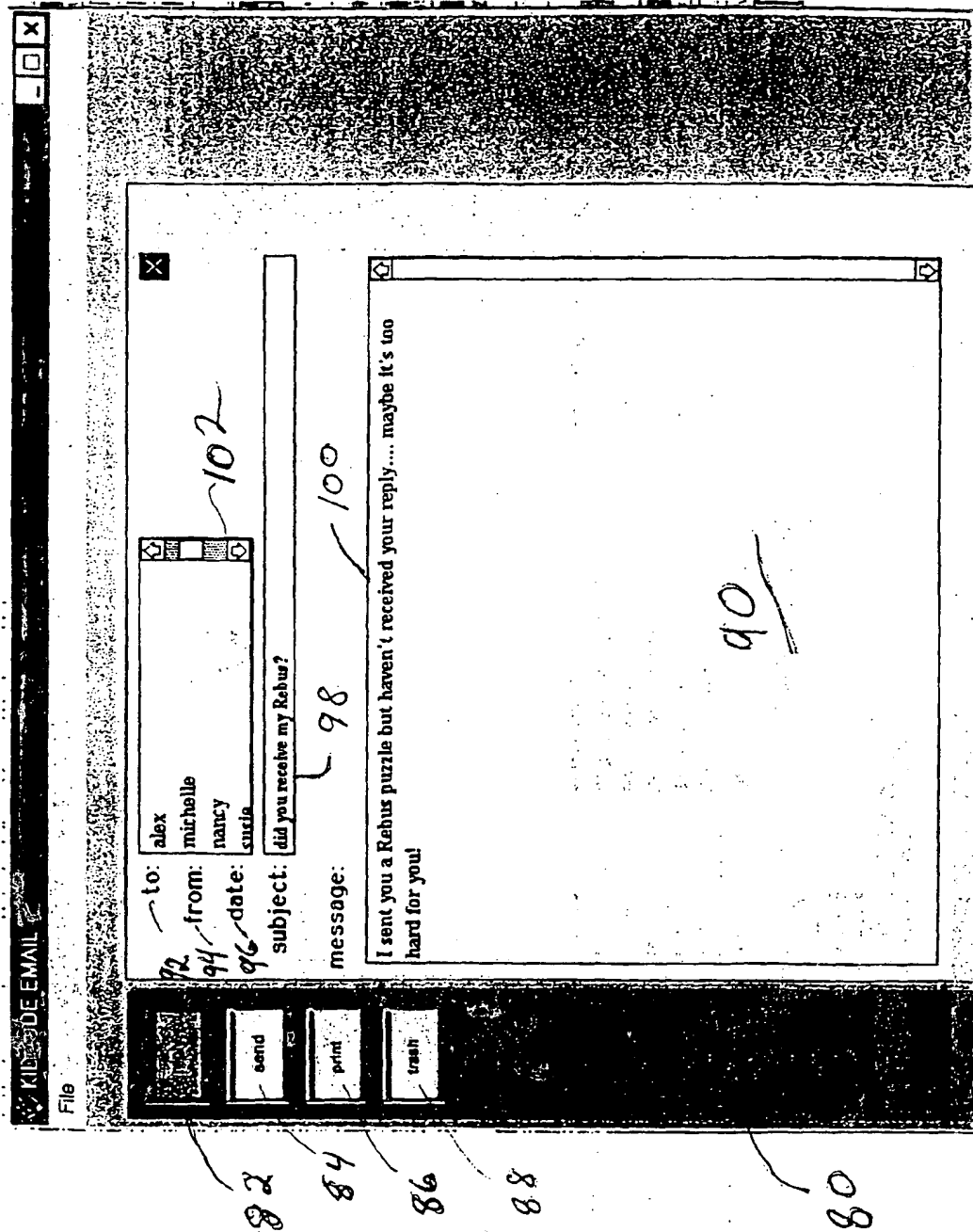
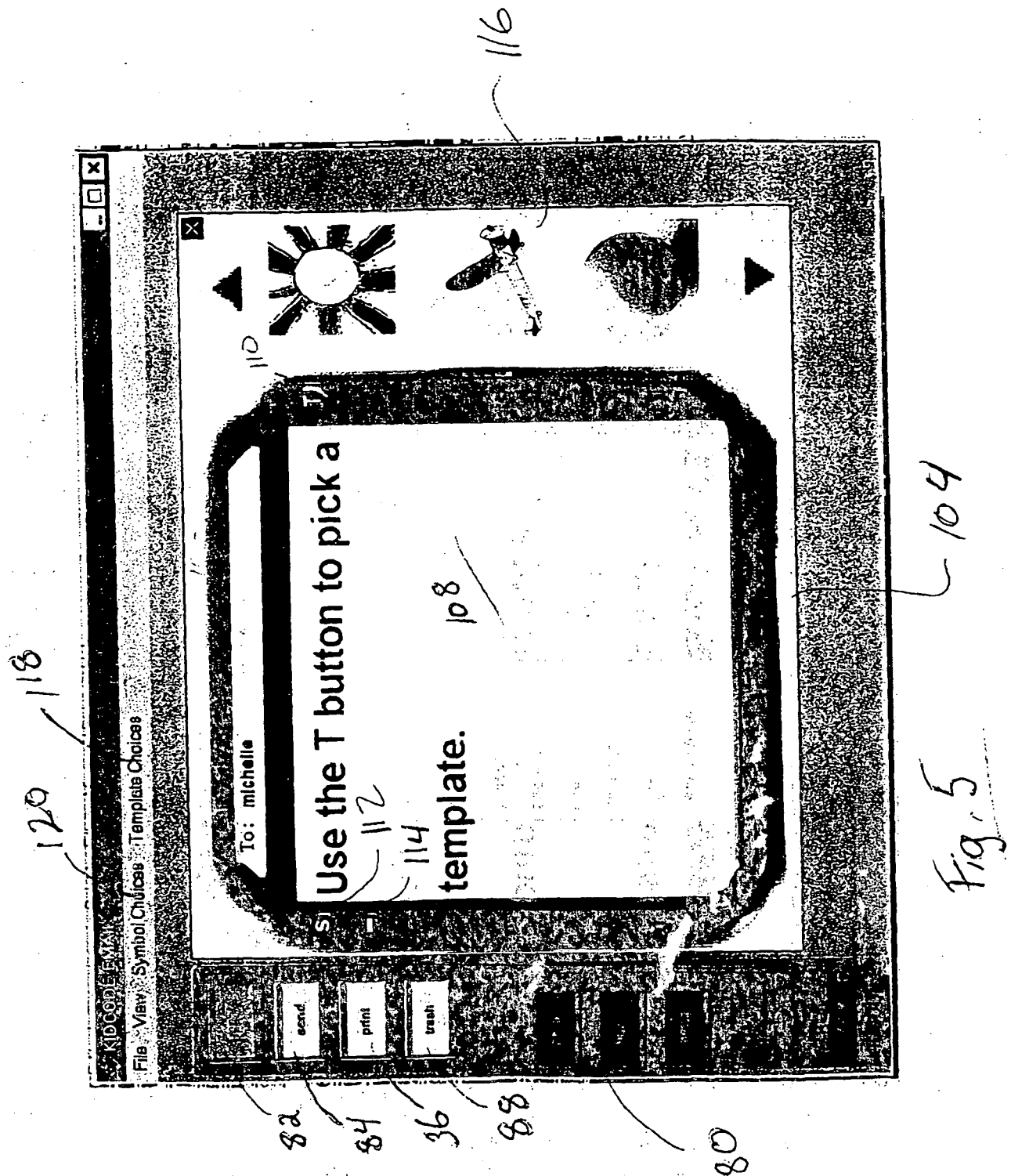


Fig. 4



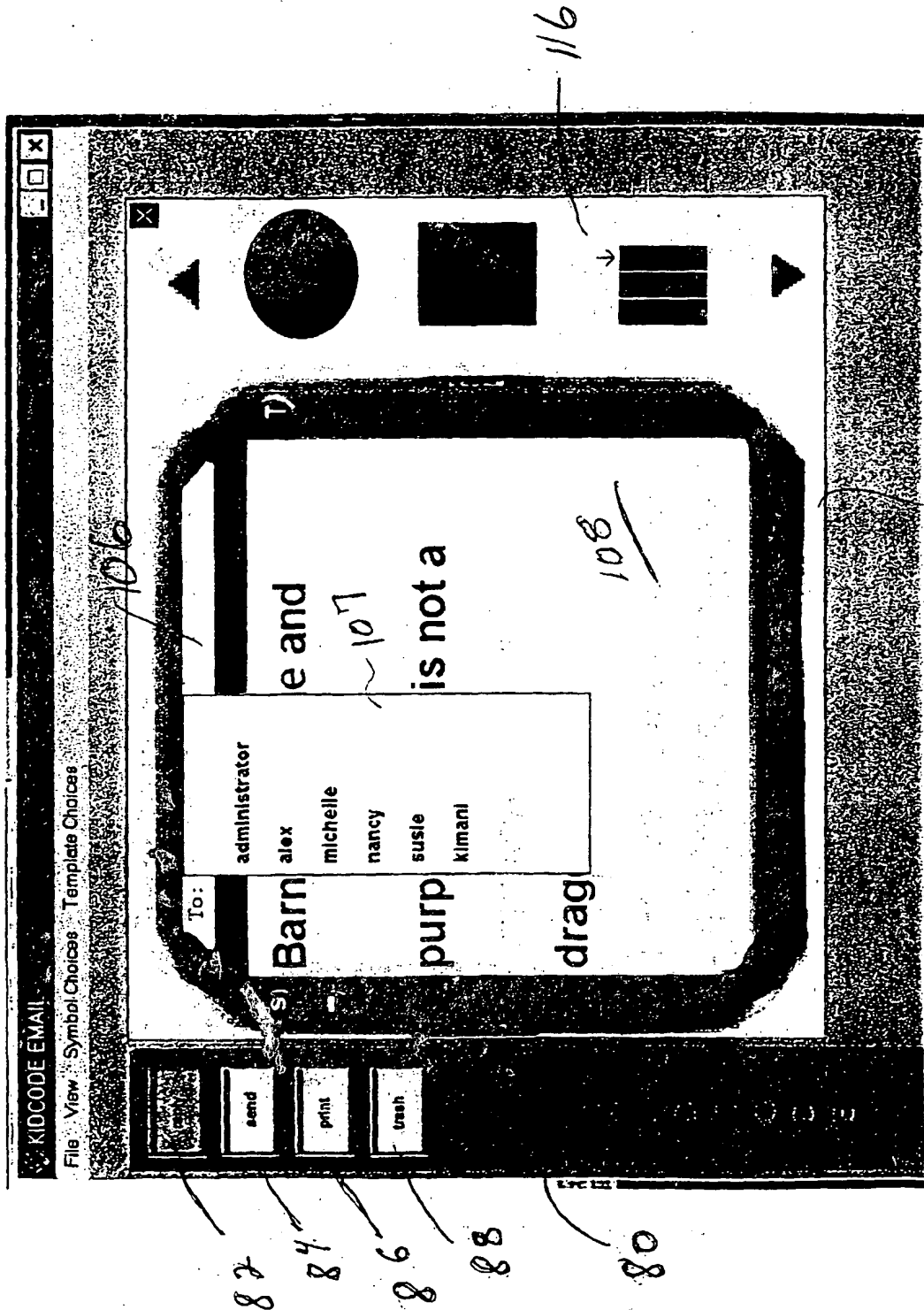
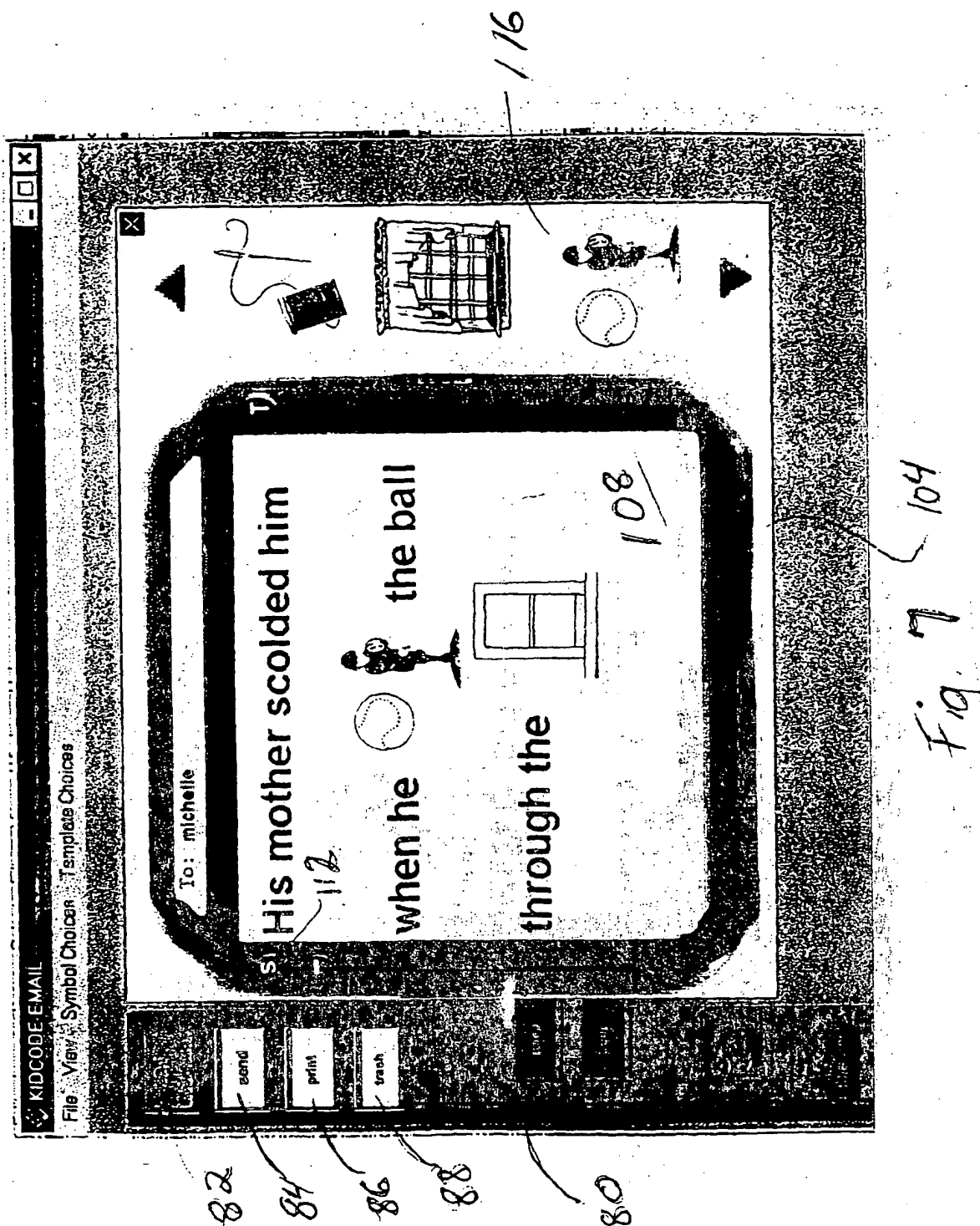
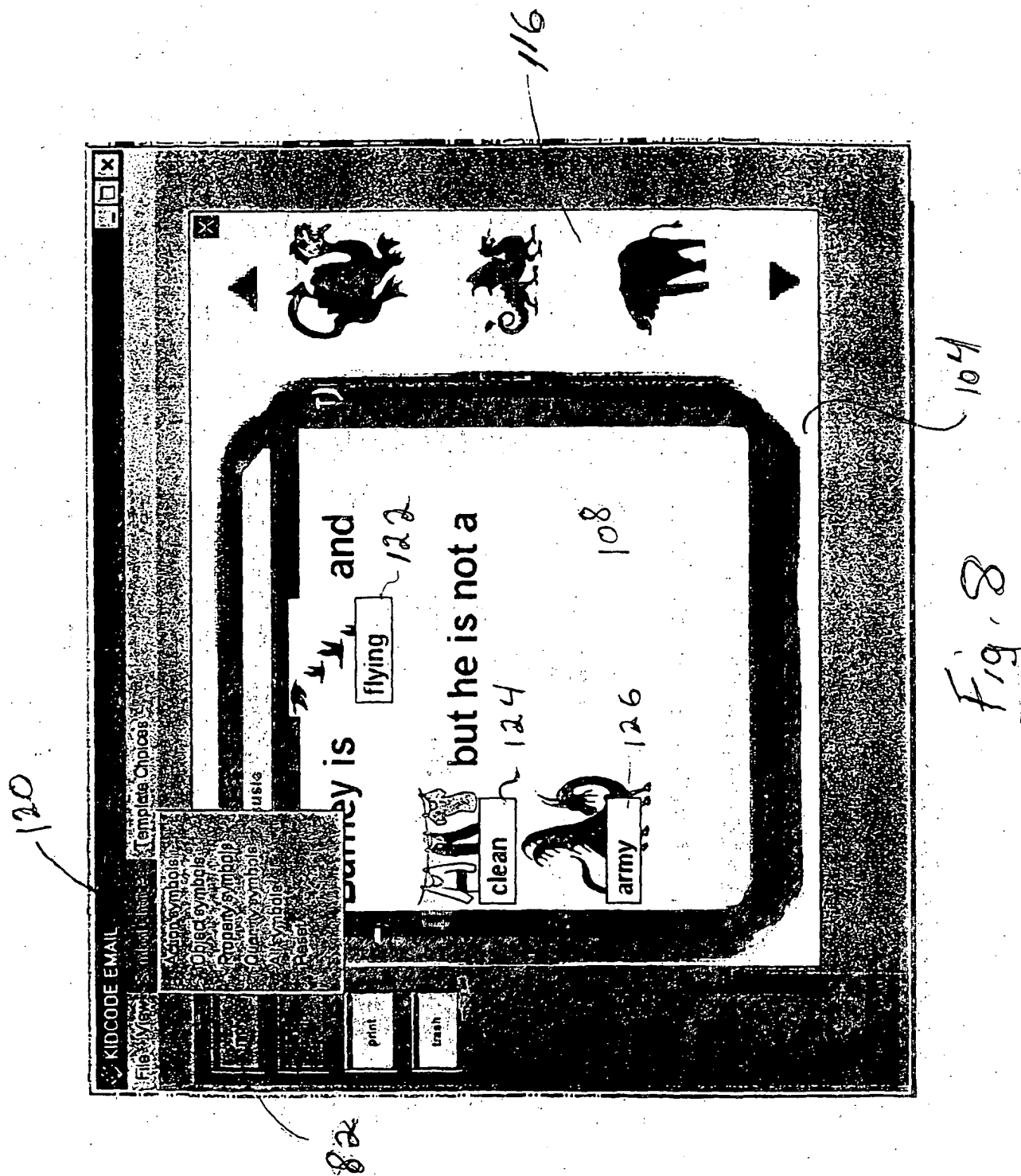


Fig. 6





KIDCODE EMAIL

File

send

print

trash

Daily Problems

Monday September 15

130

to:

from:

132

134

152

136

140

142

144

146

Alphabet Code

a = 1	n = 14
b = 2	o = 15
c = 3	p = 16
d = 4	q = 17
e = 5	r = 18
f = 6	s = 19
g = 7	t = 20
h = 8	u = 21
i = 9	v = 22
j = 10	w = 23
k = 11	x = 24
l = 12	y = 25
m = 13	z = 26

Use the alphabet code. Find three words that each have a value ≤ 30 .

148

132

134

152

136

140

142

144

146

130

Fig. 9

82

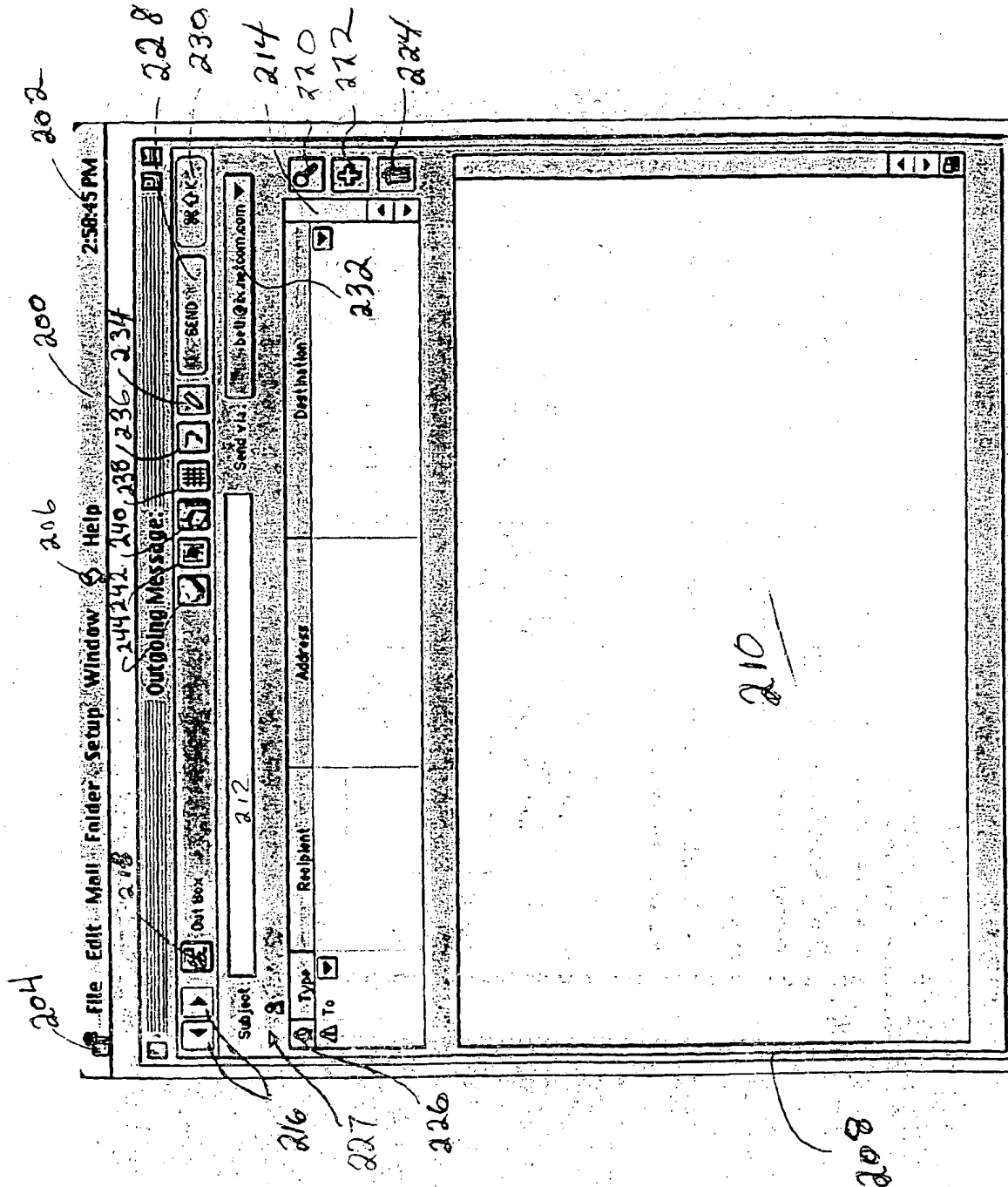


Fig. 10

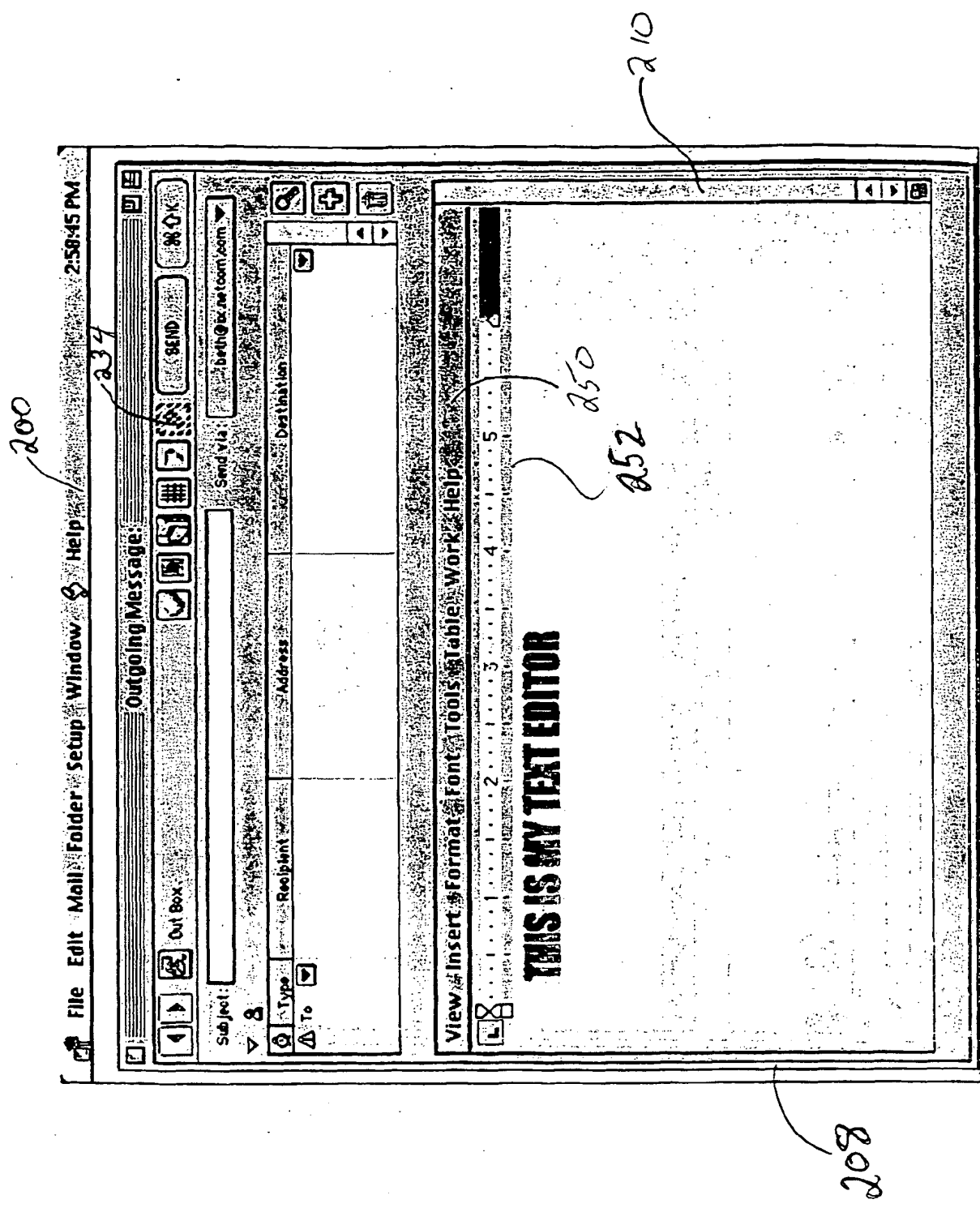


Fig. 11

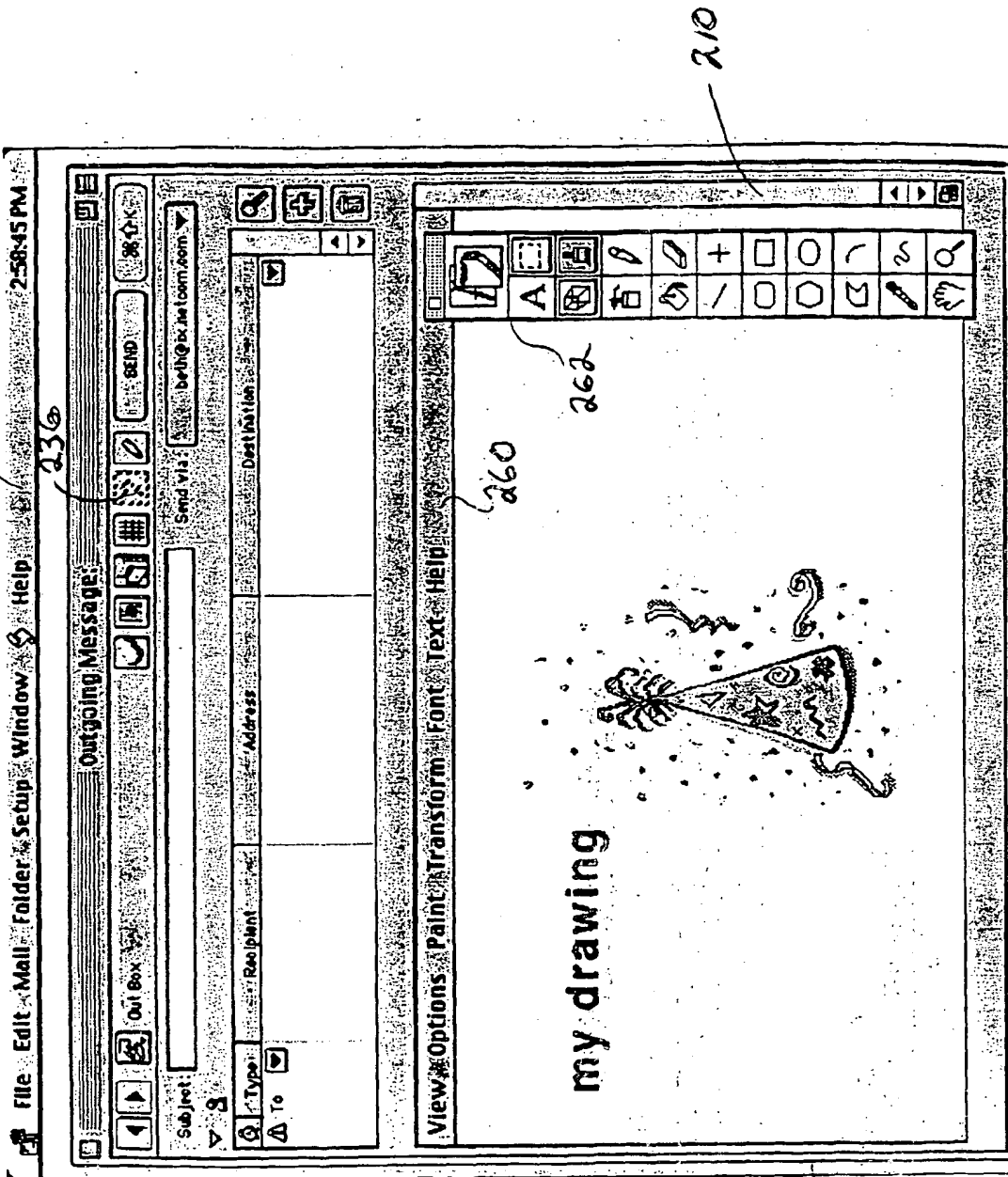


Fig. 12

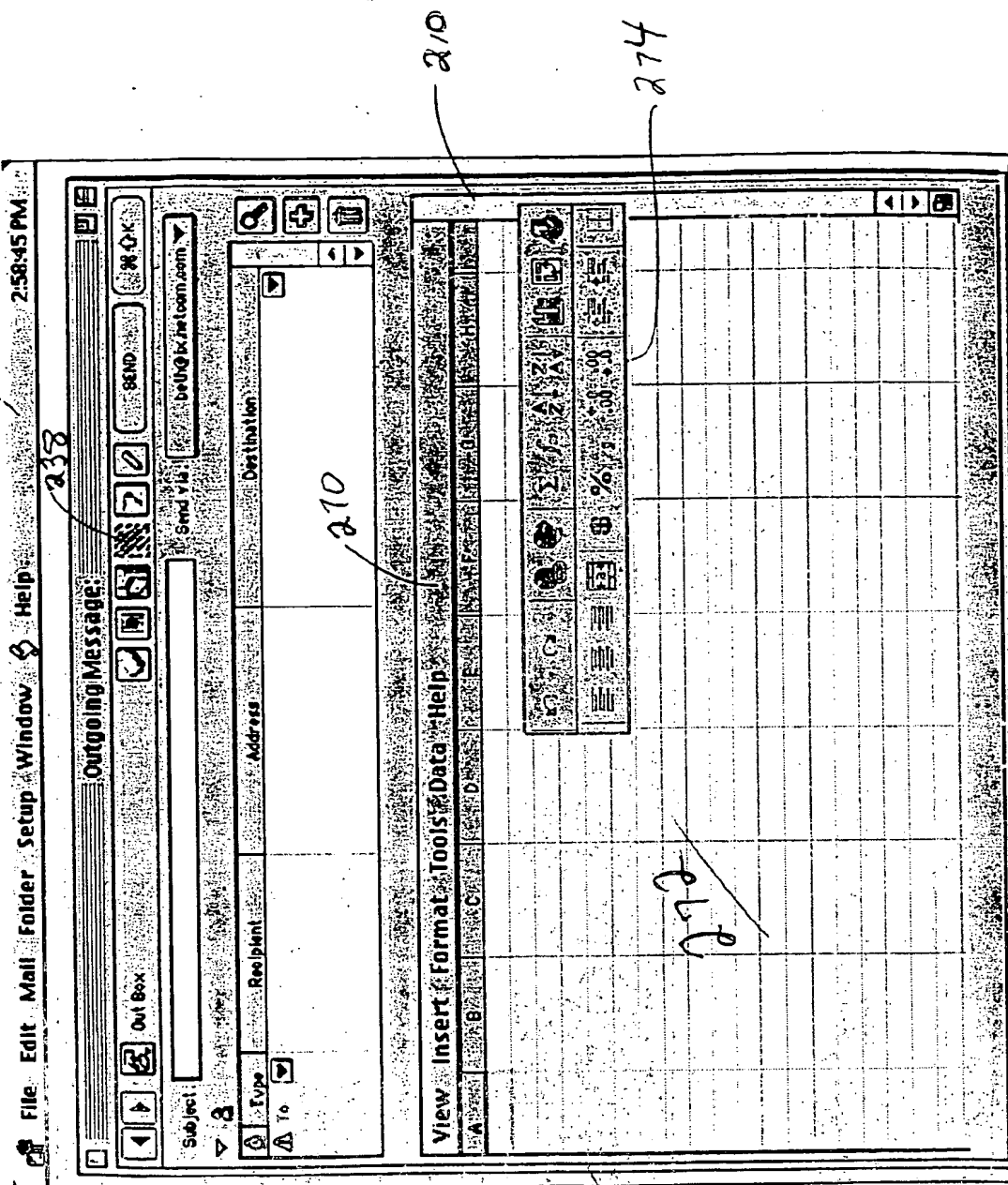


Fig. 13

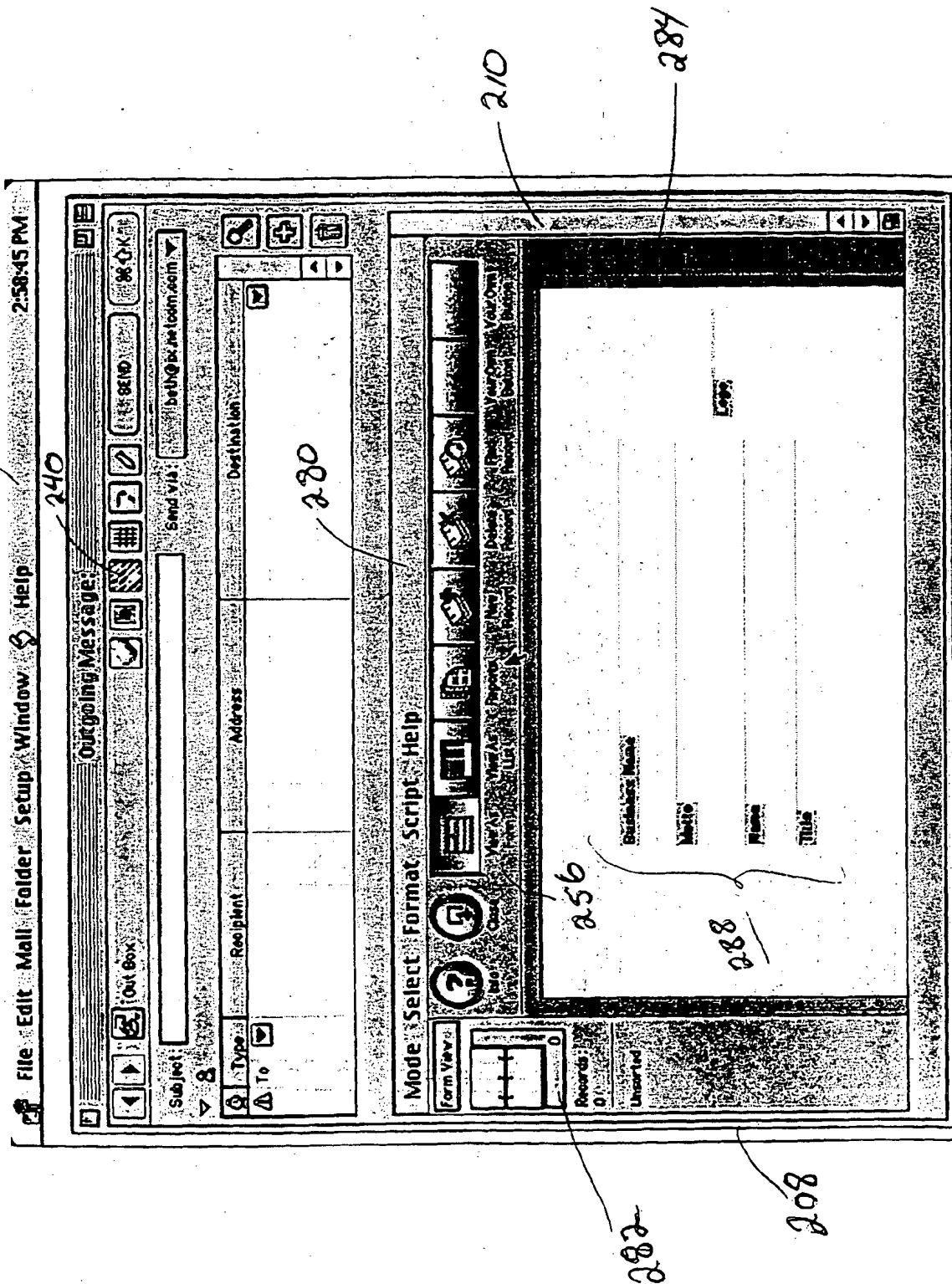


Fig. 14

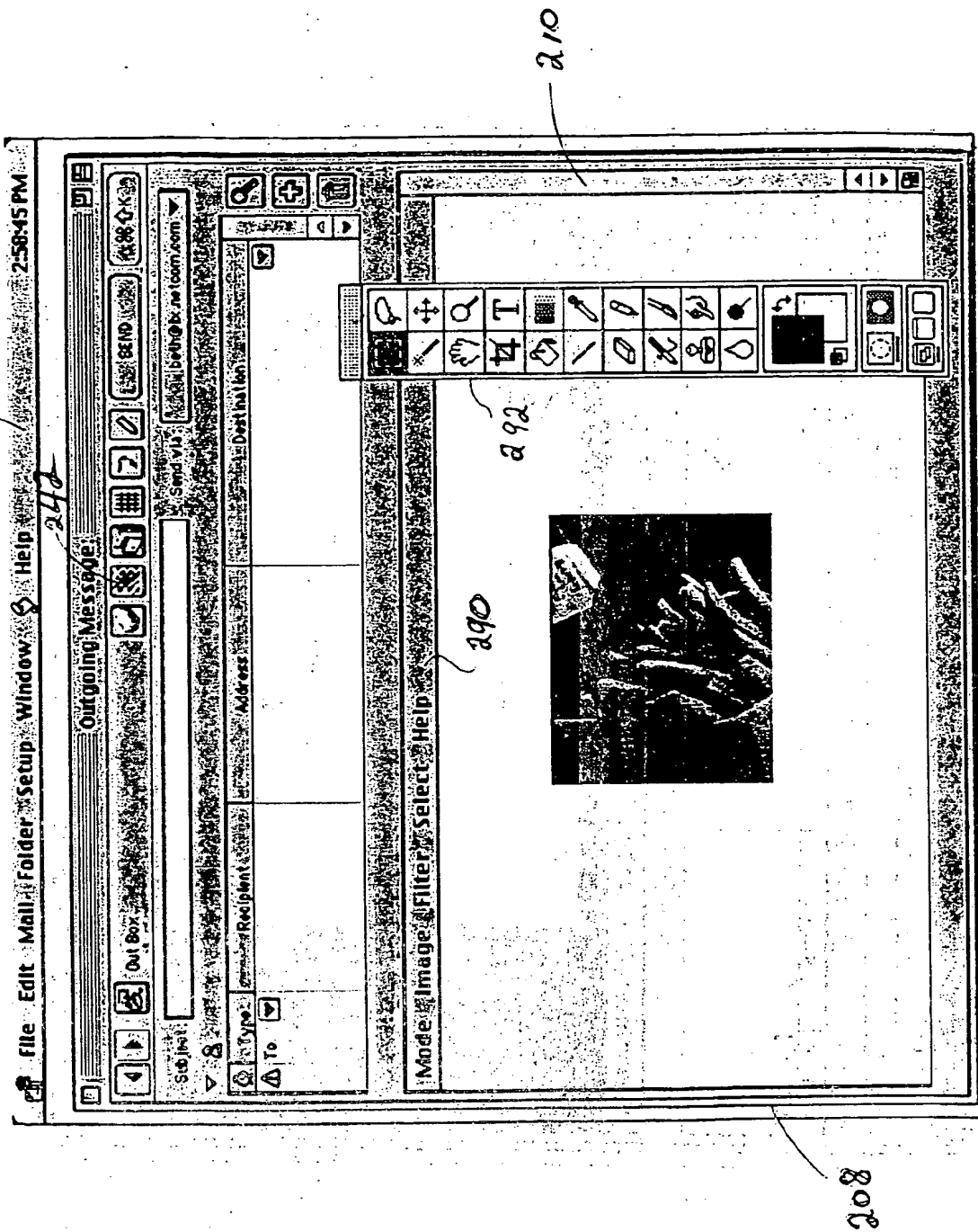


Fig. 15

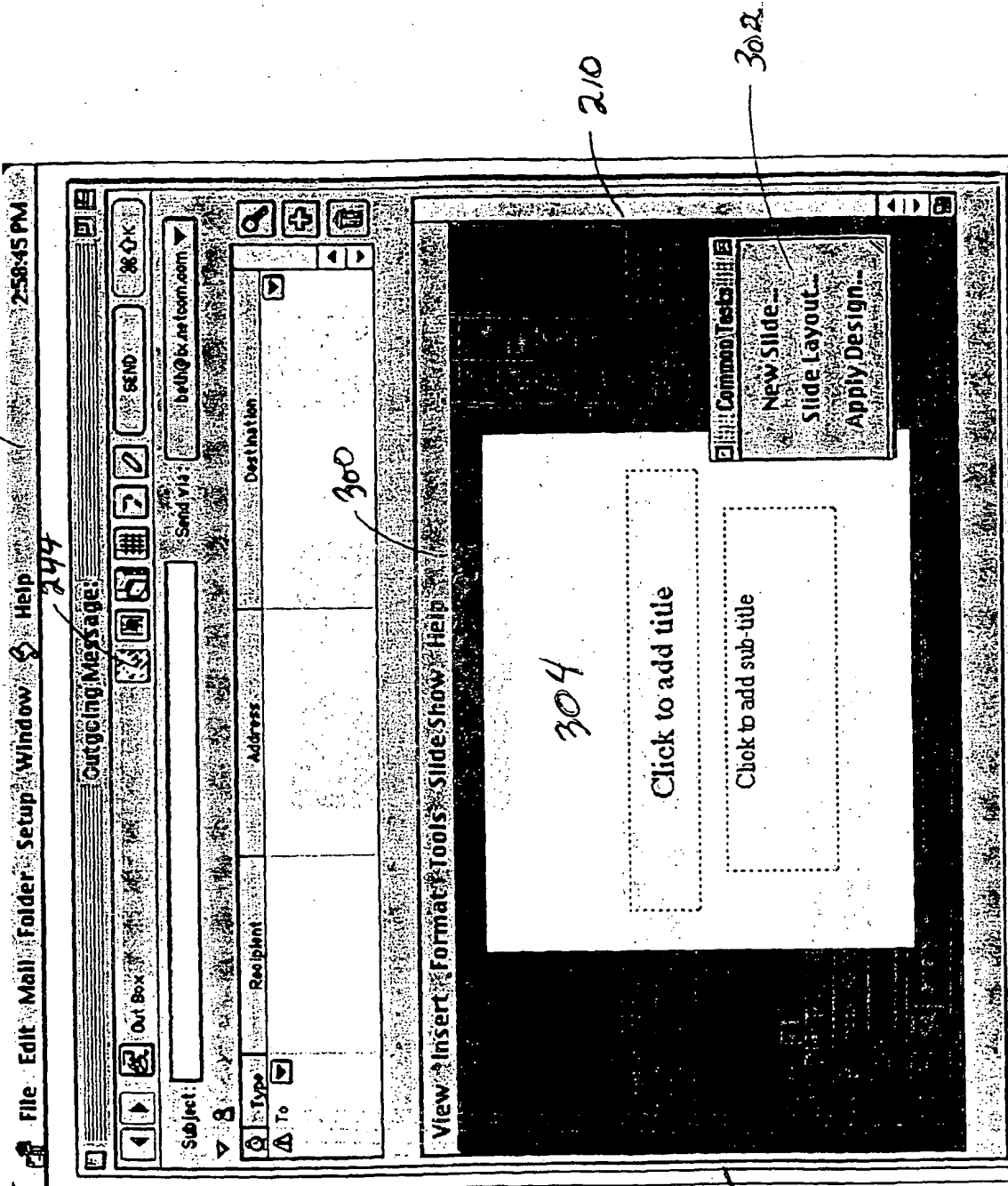


Fig. 16

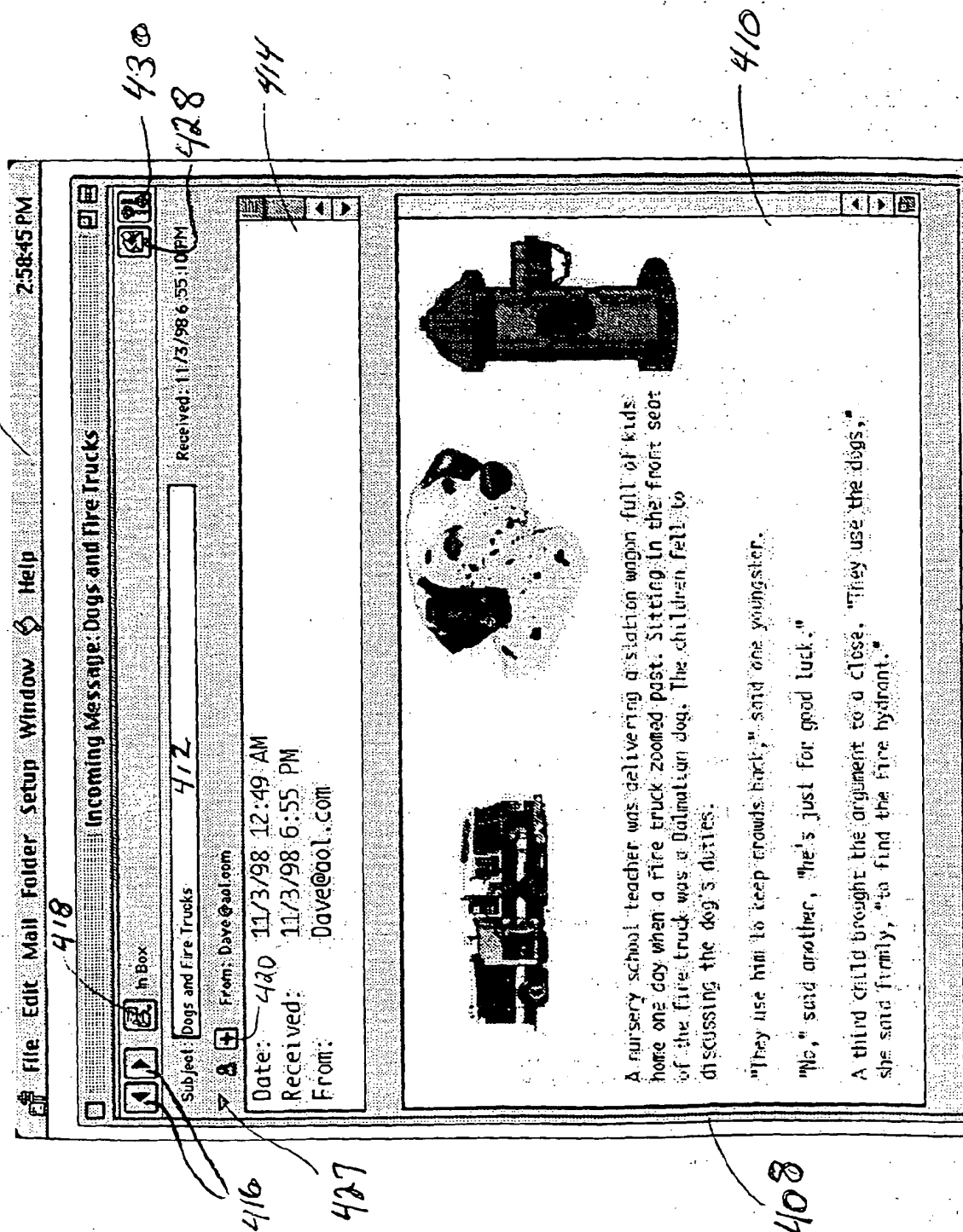


Fig 17

INTERNATIONAL SEARCH REPORT

International application No.

PCT/US99/29368

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) : G06F 3/14
US CL : 707/526, 530, 531, 709/206

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
U.S. : 707/526, 530, 531, 709/206

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
EAST, WEST, ITKNOWLEGE.COM

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A, Y	US 5,710,883 A (HONG et al) 20 January 1998, column 2-50	5-7, 17-19, 23-24
A, Y	US 5,818,447 A (WOLF et al) 06 October 1998, column 23	3-4, 8-9, 12, 15-16, 21-22
X	FLEMING, H. Internet Explorer 4 6-in-1, Macmillan Comp. Pub., pages 182-185, 201-205, 210-217, 265-271	1-2, 10-11, 14, 20
Y		3-9, 12-13, 21-24
Y	GUILDFORD, E. Netscape Communicator 6-in-1, Macmillan Comp. Pub., pages 218-223, 259-262, 264-268	3-4, 8-9, 12, 15-16, 21-22
A, E	US A 6,014,688 A (VENKATRAMAN et al) 11 January 2000, column 1, lines 50-67, column 2, lines 1-67.	3-4, 8-9, 12, 15-16, 21-22
A, E	US A 5,995,756 A (HERRMAN et al) 30 November 1999, column 3, lines 20-67.	3-4, 8-9, 12, 15-16, 21-22
A, Y	US A 5,706,434 A (KREMEN et al) 06 January 1998, column 3, lines 1-67.	3-4, 8-9, 12, 15-16, 21-22
A, P	US A 5,889,518 A (POREH et al) 30 March 1999, column 73, lines 20-67.	3-4, 8-9, 12, 15-16, 21-22
A, Y	US A 5,471,470 A (SHARMA et al) 28 November 1995, column 2, lines 20-67.	3-4, 8-9, 12, 15-16, 21-22



Further documents are listed in the continuation of Box C



See patent family annex

Special categories of cited documents:	
"A" document defining the general state of the art which is not considered to be of particular relevance	"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"E" earlier application or patent published on or after the international filing date	"X" document of particular relevance: the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"I" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"Y" document of particular relevance: the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"O" document referring to an oral disclosure, use, exhibition or other means	"&" document member of the same patent family
"P" document published prior to the international filing date but later than the priority date claimed	

Date of the actual completion of the international search

Date of mailing of the international search report

06 APR 2000

Name and mailing address of the ISA/US
Commissioner of Patents and Trademarks
Box PCT
Washington, D.C. 20231

Facsimile No. (703)305-3230

Authorized officer

STEPHEN HONG

James R. Matthews

Telephone No. (703) 305-3900

INTERNATIONAL SEARCH REPORT

International application No.

PCT/US99/29368

C (Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A, Y	US A 5,835,769 A (JERVIS et al) 10 November 1998, column 2, lines 1-56.	3-4, 8-9, 12, 15-16, 21-22
A, Y	US A 5,452,289 A (SHARMA et al) 19 September 1995, column 2, lines 8-67.	3-4, 8-9, 12, 15-16, 21-22